电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论文题目 水电企业大数据基础平台的设计与实现

专业学位类别 工 程 硕 士

学 号 201552201015

作 者 姓 名 冯 俊

指 导 教 师 郭建东 副教授

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

水电企业大数据基础平台的设计与实现

（题名和副题名）

冯俊

（作者姓名）

指导教师 郭建东 副教授

电子科技大学 成 都

熊开智 教授级高级工程师

雅砻江流域水电公司 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工 程 硕 士

工程领域名称 软件工程

提交论文日期 2018.3.15 论文答辩日期 2018.05.18

学位授予单位和日期 电子科技大学 2018 年 6 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

Design and implementation of big data foundation platform for hydropower enterprises

A Master Thesis Submitted to
University of Electronic Science and Technology of China

Discipline: **Master of Engineering**

Author: **Jun Feng**

Supervisor: **Jiandong Guo**

School: **School of Information and Software Engineering**

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。据我所知，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，也不包含为获得电子科技大学或其它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。

作者签名：冯俊　　日期：2018年 5月 29日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技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，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应遵守此规定）

作者签名：冯俊　　导师签名：

日期：2018 年 5 月 29 日

摘 要

随着水电企业信息化的不断发展，企业已经累积了大量的结构化数据和非结构化数据，同时还存在潜在的可采集的海量实时数据。现今，数据已是企业的无形资产,企业对于利用数据驱动发展的需求十分迫切。当前水电企业在建设数据中心时基本采用传统架构，存在扩展性差、建设成本高、运行成本高，支持数据类型单一、数据处理效率低下等问题。无法满足大数据时代下高速增长的全类型数据存储和处理需求，不能支撑大数据时代下水电企业深度利用潜在数据资产的需求。

本文针对目前的问题，梳理了水电企业的信息数据资源，完成了水电企业大数据基础平台的需求分析，完成了混合架构的大数据基础平台的设计和实现。大数据基础平台主包括两个数据集成层和数据存储层。

大数据基础平台的数据集成层针对水电企业各信息系统、自动化系统中的结构化数据、非结构化数据和实时数据三种类型的数据集成需求，从数据场景、技术方式、数据特征、触发机制、处理步骤等维度总结，数据集成层通过接口表、接口数据文件、接口调用、消息队列等方式，实现数据的自动收集、整理、清洗、转换，并存储到平台的数据存储层。

大数据基础平台的数据存储层包含数据仓库平台，分布式数据平台，流数据平台。基于国产数据库搭建 Gbase 8T 数据仓库平台，数据仓库进行了分区设计，分为缓冲区、整合区、汇总区和集市区，通过 ETL 技术在数据仓库内对数据实现整合、汇总，实现结构化数据按照业务主题域进行分类和存储。基于 Hadoop 搭建分布式数据平台，文件格式采用 HDFS 分布式文件系统，数据库采用 HBase 分布式列式数据库，满足海量数据存储和并发需求。分布式数据平台进行了分区设计，分为非结构化数据区，流数据转储区。非结构化数据区实现了非结构化数据的存储，并与结构化数据建立了关联。流数据转储区实现了海量实时消息数据的持久化存储。基于“Kafka+Storm+Redis”搭建流数据平台，针对数据源层中的各种实时消息进行，实现高效的，可靠的，实时的流式处理并存储。

大数据基础平台实现流域全类型数据的集中存储和整合，具备高数据处理能力，解决了企业内部各信息系统的数据孤岛情况，为后续挖掘数据价值，实现数据驱动企业打下基础。

关键词：大数据，分布式平台，流数据平台，水电企业

ABSTRACT

With the continuous development of hydropower enterprise informatization, enterprises have accumulated a large amount of structured data and unstructured data, and there are also potential massive real-time data that can be collected. Nowadays, data is an intangible asset of an enterprise. The need for enterprises to use data-driven development is urgent. At present, hydropower enterprises basically adopt traditional architectures when building data centers, which have the problems of poor scalability, high construction costs, high operating costs, support for single data types, and low data processing efficiency. It is unable to meet the demand for high-speed growth of all types of data storage and processing in the era of big data, and cannot support the needs of hydropower enterprises for the deep use of potential data assets under the era of big data.

This thesis aims at the current problems, sorts out the information data resources of hydropower enterprises, completes the analysis of the needs of hydropower enterprises' big data foundation platforms, and completes the design and implementation of the big data foundation platform of hybrid architecture. The main platform of big data includes two data integration layers and data storage layers.

The data integration layer of the big data foundation platform addresses the three types of data integration requirements for hydropower enterprises' information systems, structured data in automated systems, unstructured data and real-time data, from data scenarios, technical methods, data features, and trigger mechanisms. The dimension of the processing steps is summarized. The data integration layer automatically collects, sorts, cleans, converts, and stores data to the data storage layer of the big data base platform through interface tables, interface data files, interface calls, and message queues.

The data storage layer of the big data foundation platform includes a data warehouse platform, a distributed data platform, and a streaming data platform. Based on the domestic database to build the Gbase 8T data warehouse platform, the data warehouse was partitioned and divided into Origin Data Model, Foundation Data Model，Aggregation Data Model, Mart Data Model. The data was integrated and summarized in the data warehouse through ETL technology to realize structured data. Classify and

store according to the business subject domain. Based on Hadoop's distributed data platform, the HDFS distributed file system is used as the file format, and HBase's distributed column database is used as the database to meet the massive data storage and concurrent demand. The distributed data platform is partitioned and divided into unstructured data areas and streaming data dump areas. The unstructured data area implements the storage of unstructured data and is associated with structured data. The stream data dump area implements persistent storage of massive real-time message data. Based on the "Kafka+Storm+Redis" platform to build stream data platform, it can perform various real-time messages in the data source layer to achieve efficient, reliable, real-time streaming processing and storage.

The Big Data Foundation Platform realizes centralized storage and integration of all types of river basin data, has high data processing capabilities, resolves the situation of data islands in each enterprise's internal information systems, and lays a foundation for subsequent data mining and data driven enterprises.

Keywords: Big data,Distributed data platform,Streaming data platform, Hydropower Enterprises

目 录

第一章 绪论

1.1 课题背景及研究意义

1.1.1 选题背景

雅砻江流域水电开发有限公司大型水力发电企业，负责雅砻江流域梯级水电站的建设，开发利用流域水力资源。为科学开展雅砻江水力开发，统筹管理流域电站，迈入国际一流发电企业行列，适应未来电力市场变化的需求，公司战略明确，结合当前大数据、云计算、移动应用、物联网等多方面信息化技术，进一步提高公司流域开发管理水平，提质增效。

在公司战略指导下，公司在信息化规划中明确了建设企业数据中心和运监与决策中心，利用大数据在海量数据存储，并行计算，挖掘分析等方面的能力，为企业提供流域全类型的数据服务。本文作者负责其中子系统大数据基础平台的需求分析、设计及部分实现工作。

1.1.2 选题的意义

现今，数据已是企业的无形资产，驱动着企业发展。对数据采集存储后，通过数据分析，提高企业运营效率，为企业运营提供决策，为企业战略提供支持，从而驱动着企业发展。

随着公司业务的不断展开,信息化程度不断提升，公司信息化系统已经累积了大量的结构化数据和非结构化数据，同时还存在潜在的可采集的海量实时数据[1]。公司对于利用数据驱动发展的需求十分迫切。然而目前公司系统基本采用传统架构，如 Oracle Rac[2]，存在扩展性差、成本高、支持数据类型单一、数据处理效率低下等问题，无法满足全类型数据高速增长的存储要求，无法满足大数据量的高速处理需求，无法满足大数据环境下全类型数据的存储和计算需求，无法为企业数据资产深度利用提供支撑[3]。

本项目将建立的大数据基础平台，就是为了解决传统架构存在的问题，使数据中心具备扩展力，支持全类型数据，全面提高数据处理能力。并将企业数据持续的接入大数据基础平台，集中存储，为后续企业数据资产挖掘打下基础[4]。

1.2 国内外研究现状

随着数据时代的不断发展，数据对企业发展的价值被越来越多的人所认识。

在信息化发展较好的行业中，存在着海量的数据，除了数据量的井喷增长外，数据类型也从原来得结构化数据类型上，新增了各种格式的图片、影音、文档等非结构化数据[5]。面对这海量的数据，多样的数据类型，如何有效的实现企业数据集成，减少数据使用成本，充分数据资源，成为了一个重要研究的问题。

世界早已开始关注和研究如何利用数据推动文明的进程。2012 年，美国发布了《大数据研究和发展倡议》，旨在促进从海量数据中获取价值的能力。同年，联合国也发布了涉及大数据的政务白皮书《大数据促发展：挑战与机遇》。在这份政务白皮书中，联合国提出了大数据时代是一个机遇，是一个全球的历史性的机遇，探讨了如何利用丰富的数据资源，更好的为社会和经济服务，推动全球发展。

我国也十分重视大数据的发展，在国家“十二五”国家战略性新兴产业发展规划中，需大数据技术支撑的信息处理技术已经是关键技术创新工程之一。在国家“十三五”国家战略性新兴产业发展规划中，直接提出了实施国家大数据战略。重点指出在大数据时代下需要高效集成数据，有效整合数据，公开共享数据，应用拓展数据，规划明确国家要利用大数据技术，建立宏观调控体系，同时推进重点行业的大数据平台建设，推进大数据在重点领域的应用。

在大数据时代的背景下，更多行业认识到数据的价值，然而数据的海量和多样的类型使得传统的关系数据库在存储能力和查询性能上都难以满足需求[6]。各行业开始构建相应大数据模型，建设企业大数据平台，开展大数据应用。大数据的相关的技术和设计思路开始高速发展，分布式系统基础架构 Hadoop；多种类型数据的混合存储；混合存储下的数据识别、关联、查询；内存数据库；大数据的集成、清洗、管理等技术已进入到大数据在各行业的实际应用中。

国外互联网行业早先一步对大数据应用开始了研究，并实际投入应用。同时金融、通信、能源等行业也使用大数据技术，用于探索数据价值，创新业务，提高企业竞争力。

国内在大数据时代也并未落后，绝大多数重要行业已经开始探索大数据的应用，数据的意义已经得到全面认可，数据的实际回报已经初见苗头，大数据在国内大有可为。

互联网行业中，面对每日可达 TB 甚至 PB 级的海量的用户信息，如 Facebook、Twitter、亚马逊、阿里、腾讯等国内外各大型互联网企业，都建立了自己的大数据中心，对数据进行处理、存储和计算。分析用户全过程的互联网行为数据，给出精准的搜索、推送、投放广告，获取全网热点数据，热门趋势，为企业发展服务。谷歌公司通过大数据技术，就为美国经济带来千亿美元的收入，这也只是大数据技术为美国经济带来收益的情形之一[7]。

互联网企业同时利用大数据平台对外开展了大数据服务，阿里的大数据平台开始集成全网的消费类数据，金融类数据，开展建设 Data Exchange Center，将数据作为商品进行交易，体现数据潜在的价值。新浪的企业微博，也对用户开展数据分析服务。百度开展为用户提供基于其搭建的数据体系平台得到的数据分析而来的消费者行为，市场的动态，行业的发展趋势等信息。

通信行业，传统关系型数据库无法满足现今 PB 级别的通联记录量的数据，通信行业同样建设了大数据平台，实现对电信数据的存储和处理。

金融行业除了数据存储后离线分析外，在线流式数据计算应用也发展迅速。金融系统中，日常交易生成海量的数据在各系统，各企业间流动，流式数据的实时分析，对金融智能决策，风险管控，实时预警提供支持。

在物联网应用领域，各类传感器生成海量数据[8]，这些数据具备多样性，实时性等特点，利用大数据的对数据进行存储后分析和实时的在线分析，有着极大的意义。

在电力行业，信息化也已覆盖几乎所有环节，各类信息系统和自动化系统中存在的海量数据的存储和计算已是极大的需求[9]。国家电网公司早已经设计了自己的企业数据模型，建立了数据中心，实现各企业内部各信息系统的数据集成和数据共享，实现了电网公司的数据统一，对企业数据质量进行治理。在大数据时代的背景下，电网公司开始开展了智能电网的应用，引入分布式存储计算平台，结合关系型数据库，解决数据应用的性能瓶颈问题[10-12]，通过获取用户用电行为、企业内部数据、设备物联网数据的集成，支撑精细化客户管理；支撑电网安全生产和控制；支撑电力调度决策；支撑电量需求预测和趋势分析；支撑电力企业智能决策，充分利用大数据获取电力企业数据中的价值，实现数据驱动电力企业发展。

1.3 论文主要研究内容

（1）梳理公司信息数据资源

完成各个业务系统数据资源的梳理，完成数据接入需求分析，包括指标涉及的业务系统范围，指标数据范围、实施指标计算要求等，得出数据范围、格式、更新方式、更新频率、质量等各方面分析结果。以数据源分析的结果为指导，完成公司大数据基础平台需求分析。

（2）大数据基础平台系统设计与实现

根据需求分析，完成设计大数据基础平台的总体架构设计，形成各类源业务系统数据接入的方案和数据存储方案。

完成大数据基础平台数据集成模块的设计与搭建，通过多种采集手段，构建企业级统一的数据采集服务，实现对雅砻江公司结构化数据、非结构化数据、实时数据等不同类型场景数据的集中接入，实现业务系统和大数据基础平台数据的集成数据集成。

完成大数据基础平台数据存储平台的设计与搭建，对数据的物理层进行设计，对数据的存储方式进行设计，确定数据的存储结构，确定索引策略，确定数据存放位置，确定存储分配等。搭建数据仓库平台、分布式数据平台、流数据平台 3 个数据存储平台，实现了结构化、非结构化、实时数据的存储，将结构化数据的存储到数据仓库平台，非结构化数据及海量结构化数据存储到分布式数据平台，实时数据存储到流数据平台。

1.4 论文组织结构

本论文各章节结构安排如下：

第一章：绪论。主要介绍课题背景和选题意义，国内外在大数据平台建设方面的现状。介绍了本论文的主要研究内容和论文组织结构。

第二章：相关技术。介绍系统建设的采用的 Kettle，Hadoop，HBase，Storm，Kafka，Redis，GBase 等技术。

第三章：系统需求分析。介绍雅砻江水电流域开发有限公司信息系统现状，数据现状，对大数据基础平台系统进行了比较详细的需求分析。

第四章：系统设计。介绍了大数据基础平台设计目标，详述了平台整体架构，各类数据集成场景和各类数据存储平台的设计。

第五章：系统实现与测试。介绍了大数据基础平台的硬件环境，详述了平台大数据基础平台的搭建以及模块的具体实现，对大数据基础平台进行了相关测试。

第六章：总结与展望。总结了大数据基础平台的建设工作及成效，并对后续大数据基础平台的完善及大数据的分析工作进行了展望。

第二章 相关技术

2.1 ETL 工具 Kettle

Kettle 是基于 Java 语言开发的 ETL 工具，已是国内外主流的 ETL 工具，可在如 Windows 和 Linux 等主流操作系统上使用。Kettle 具备直观易用的图形化界面，提供脚本使我们方便的自定义数据的转换，监控任务的流程，并且具备批处理能力。实现了异构数据稳定高效的抽取，转换和加载，高性能的集成数据，优化数据[13]。在本系统中，Kettle 将主要用于关系型数据的抽取、转换和加载。

2.2 大数据架构 Hadoop

随着海量的非结构化数据存储需求的增长，传统的数据库架构已不能满足现今数据需求的发展。分布式系统应运而生，Hadoop 是其中一种分布式系统基础架构，其由 HDFS（Hadoop Distributed File System）、MapReduce 和 YARN 组成[14-15]。HDFS 实现海量数据的分布式存储，MapReduce 实现 HDFS 中的存储数据的分布式计算[16]，YARN 实现海量数据运算时的资源调度，其基于分布式架构，可部署在低成本的硬件设备上，具备高容错率，高并发实现高吞吐[17]。HDFS 使用名称节点管理元数据，数据节点存储数据块，只需增加数据节点，便能实现系统横向扩容[18]。在本系统中，应用 Hadoop 架构中的 HDFS 为分布式列式数据库 HBase 提供底层存储，YARN 提供了资源调度的功能。

2.3 分布式列式数据库 HBase

HBase 是一个开源数据库，其与传统的关系型数据库不同。传统的关系型数据库面向行，HBase 面向列；传统的关系型数据库集中式，HBase 为分布式[19]。HBase 是 Hadoop 项目中重要的一个子项目，适用于海量数据存储，海量数据的高并发作业，随机但简单的读写海量数据等场景[20]。其表可支持百万列，上亿行；以列为单位，面向列的存储和检索[21]；Null 不消耗存储空间，表设计支持稀疏性，表可看作一个稀疏行集合，但物理存储上，按列分开存储表[22]；数据类型均为 String；数据存储支持多副本。整个 Hadoop 体系还为 HBase 提供了支持高层语言的工具，实现简单化的数据处理；提供了方便的数据迁移工具，使传统的关系型数据库数据可十分便利的迁移至 HBase，同时也可将 HBase 中的数据迁移至关系型数据库中[23-24]。在本系统中，分布式列式数据库 HBase 提供了海量数据高效存储、查询

的功能。

2.4 流计算框架 Storm

Storm 是一个分布式实时计算框架，是 Apache 的重要的开源项目。Storm 具备高容错性和高可靠性，自身的机制实现对节点和任务的管理，保障每一个收到的数据都能得到有效的处理；Storm 具备高效率，高扩展性，其基于分布式架构，数据的处理在多服务器、多进程、多线程中并发操作，并可以简单实现节点的水平动态扩展。Storm 适用于流处理，可以高效的，可靠的，实时的处理不间断接收到的数据，并写入存储；Storm 处理数据的基本单位是元组，元组在 Spout、Bolts 中进行传递和处理，保证了数据处理的时效性，可以对实时数据进行实时的连续的计算，并对计算结果实现实时的反馈[25]。Storm 和 Hadoop 集成可形成良好的流数据处理平台[26]。在本系统中，流计算 Storm 主要用于实时数据的计算处理。

2.5 分布式消息队列 Kafka

Kafka 是一个分布式的发布订阅消息系统。Kafka 为分布式结构，支持水平动态扩展,具备高吐吞率，基于时间复杂度 O(1)的方式支持海量数据的持久化[27]。Kafka 支持消息分区，分布式消费，保障消息的顺序处理，支持离线和实时数据处理。其可由多台服务器组成多个 Broker，每条发布到 Kafka 集群的消息都有一个类别，这个类别被称为 Topic，发送消息的称之为 Producer，消费消息的称之为 Consumer[28]。本系统中，分布式消息队列 Kafka 主要用于实时数据报文的临时存储，供流计算 Storm 提取数据进行加工。

2.6 内存数据库 Redis

Redis 是一个开源的内存数据库，基于 ANSI C 编写。Redis 是 NoSQL 的[29]，数据结构为 Key-Value 形式，支持 String、List、Set 等多种数据类型[30]。Redis 的操作均为不可分割的原子性操作，并且 Redis 支持多操作的原子性。Redis 数据库的数据均存储在内存中，这使 Redis 拥有极高的性能，但因为内存的限制，不适用于海量数据的处理，Redis 适用于数据量较小，处理性能要求高的场景[31]。在本系统中，内存数据库 Redis 主要用于热点数据的临时存储，以满足后端应用的查询需求。

2.7 国产关系型数据库 GBase

GBase 8t 是南大通用公司发布的国产数据库，是一种事务型数据库，基于成

熟的 Informix 12.10 企业级数据库的授权源代码自主构。GBase 8t 采用多线程构架，可高并发的处理连接和事务，性能良好。通过 SDS 共享磁盘辅节点，HDR 高可用性数据复制等技术实现 99.999%的高可用性，满足关键业务持续运行需求[32]。GBase 8t 具备简单的图形化管理界面，通过图形化界面可便利的监控数据库运营状态，分析数据库运行性能，查看日志等。GBase 8t 的事务的处理能力与同期 Oracle 相当，在 90%以上的场景里可以代替 Oracle。在本系统中，关系型数据库 GBase 为数据仓库提供了存储。

2.8 本章小结

本章对水电企业大数据基础平台中所涉及的主要技术进行了简单的介绍，支撑后续水电企业大数据基础平台的技术选型和技术路线拟定。

第三章 系统需求分析

3.1 信息系统及数据现状

随着公司业务的不断展开、建设规模不断扩大、公司建设了工程管理系统、电力生产管理信息系统、人力资源系统、财务系统、预算系统、计划管理系统、大坝安全信息管理系统、大坝施工质量实时监控系统、集控中心数据交换平台系统、锦西电厂实时监测数据、工程文档管理系统、生产文档管理系统、档案管理系统、统一采购平台、流域三维可视化与信息集成展示与会商平台等涵盖生产、工程、财务、人资、综合管理、设备监控各个方面的信息系统。经过对各系统数据资源进行梳理，数据现状如表 3-1 所示：

表 3-1 信息系统数据资源现状

序号	系统名称	接入数据情况
1	工程管理系统	**结构化数据** **数据内容：**概算、投资计划、项目、合同、合同清单项、扣款、结算、物资、验评、计量、报量、施工安全、工程物资、UCM 文档描述等结构化数据。 **数据量情况：**数据条目数约 110 万条，最大表数据量约 14 万条。
2	电力生产管理信息系统	**结构化数据** **数据内容：**设备、项目、物资、采购合同、库存、运行值守、工单、操作票、工作票、动火票、领料单、流域调度、生产安全等结构化数据。 **数据量情况：**数据条目数约 2300 万条，其中物资结存表数据量约 2000 万条，该表每天增长量约 4 万条。 **非结构化数据** **数据内容：**工单、工作票、动火票、安全检查等附件信息。 **数据量情况：**数据量约 1G。

序号	系统名称	接入数据情况
3	人力资源管理系统	**结构化数据** **数据内容：**人员、组织、职务等。 **数据量情况：**数据条目数约 2 万条。
4	财务管理系统	**结构化数据** **数据内容：**合同、总帐、会计科目、发票、付款、资产、折旧、客户、收款、预算、核算等。 **数据量情况：**数据条目数约 380 万条。
5	预算系统	**结构化数据** **数据内容：**预算结构及金额
6	大坝安全信息管理系统	**结构化数据** **数据内容：**监测仪器、监测分量、测点、测点测值、巡检记录等结构化数据。 **数据量情况：**数据条目数约 3100 万条，其中数据量在 10 万条以上的表有 3 张，分别为测值表约 3000 万条（测值表中一个测点监测多个分量，按照一个测点 4 个分量计算，数据量约 1.2 亿条）、考证信息表约 15 万条、泄洪振动监测表约 60 万条。 **非结构化数据** **数据内容：**标准规范、技术资料、安全管理、中心报告、文件资料等非结构化数据。 **数据量情况：**文档个数约 2000 个，文档总数据量大小约 8G。
7	大坝施工质量实时监控系统	**结构化数据** **数据内容：**结构化数据：填筑碾压、上坝运输、运输加水、灌浆工程等成果性数据
8	集控中心数据交换平台	**结构化数据** **数据内容：**电能量数据、水调数据、机组状态集控数据。

序号	系统名称	接入数据情况
9	锦西电厂实时数据交换平台	**实时数据** **数据内容：**锦西电厂计算机监控系统数据。 **非结构化数据** **数据内容：**状态监测和分析系统、故障录波系统数据。
10	工程文档管理系统（UCM）	**非结构化数据** **数据内容：**质量安全文档、工程设计图纸等。 **备注：**文档的结构化描述信息存储在工程管理系统（现场）中。目前两河口、杨房沟、桐子林 3 个管理局分别部署一套工程文档管理系统，需针对各个管理局实施数据接入。 **数据量情况：**总数据量约 50G，其中两河口约 20G，杨房沟约 15G，桐子林约 13G。
11	生产文档管理系统（FILENET）	**结构化数据** **数据内容：**文档目录结构、文档描述信息等。 **非结构化数据** **数据内容：**各电厂标准规范、技术资料、技术文件、总结报告等文件。 **数据量情况：**文档个数约 5.5 万个，文档总数据量大小约 50G；2014 年增长量约 6000 个文档，大小约为 7G。
12	档案管理系统	**结构化数据：** **数据内容：**全宗数据、档案类型数据、目录树数据、档案分类数据、档案卷数据、档案基本信息数据、电子文件索引数据等。
13	统一采购平台	**结构化数据** **数据内容：**采购计划、招标信息、合同审批、合同编制、合同变更和合同结算。 **数据量情况：**数据条目数约 2 万条。

序号	系统名称	接入数据情况
14	流域公共安全信息管理与决策支持系统	**结构化数据** **数据内容：**公共安全重点保护区域及相关信息、警示设施信息。 **非结构化数据** **数据内容：**保护对象影像、警示牌影像。
15	流域征地移民信息管理与决策支持系统	**结构化数据** **数据内容：**法律法规文档结构化描述信息、征地移民信息数据和项目土地管理数据等。 **非结构化数据** **数据内容：**法律法规文档。
16	流域环保水保信息管理与决策支持系统	**结构化数据** **数据内容：**环保水保设施信息管理及运行监控数据、环保水保监测数据及专题调查管理数据、环保水保文档描述信息、环境影响和效益后评价数据。 **非结构化数据** **数据内容：**环保水保信息文档。
17	电力营销系统	**结构化数据** **数据内容：**售电结算数据。 **数据量情况：**数据条目数约 1200 条。
18	锦屏一级设备状态评价系统	**结构化数据** **数据内容：**水轮机、发电机状态评价报告、风险评估报告数据

3.2 需求分析

3.2.1 总体需求分析

根据公司采用大数据技术以利用企业数据资产驱动企业发展的规划， 首先是解决企业内部各信息系统之间数据孤岛的情况。针对公司的信息系统和数据现状，公司需求建设一个扩展性强，可支持公司全类型数据接入和存储，具备高效数据

处理能力架构的大数据基础平台。实现公司各信息系统中结构化数据、非结构化数据和实时数据的收集、整理、清洗、转换，并接入到数据存储平台，消除数据孤岛，实现数据的集中存储，满足公司流域全类型数据整合需求。以此大数据基础平台为基础，后续可以实现企业数据的集成与分析应用，为企业运营提供决策支撑，数据深化应用公司数据。

3.2.2 数据集成需求分析

根据对公司的数据梳理，数据需接入大数据基础平台的系统有：工程管理系统、电力生产管理信息系统、人力资源管理系统、财务管理系统、预算系统、大坝安全信息管理系统、大坝施工质量实时监控系统、集控中心数据交换平台、锦西电厂实时数据交换平台、工程文档管理系统（UCM）、生产文档管理系统（FILENET）、档案管理系统、统一采购平台、流域公共安全信息管理与决策支持系统、流域征地移民信息管理与决策支持系统、流域环保水保信息管理与决策支持系统、电力营销系统、锦屏一级设备状态评价等 18 个系统。

根据数据集成需求规范数据接入标准，完成公司几种类型数据接入大数据基础平台。

结构化数据接入：雅砻江非实时结构化数据包括设计期的工程规划设计基础资料和设计成果数据，施工期、运行期各业务应用系统采集的业务基础数据，以及由已有业务系统采集获得业务基础数据和分析结果数据。由于各管理系统产生经营管理数据，频度一般是日以上。

非结构化数据接入：雅砻江非结构化数据包括三大类，第一种来自于设计单位的设计基础资料和设计成果的文档和电子图档、来自于工业视频系统的音频和视频信息、来自各业务应用系统的各类文档和图片、来自互联网的网页等；第二种为水电工程项目设计人员基于三维辅助设计软件（CATIA）所建立的水电工程模型输出文件。此类数据主要定位是做备份，为将来非结构化数据分析利用做数据储备。此类数据可采用以月级的频率接入到大数据基础平台分布式存储区中；第三类为空间地理信息数据，雅砻江空间地理信息数据主要来自于专业测绘机构通过勘查并制作发布的地理信息数据，常见的地理数据包括数字高程模型（DEM）和数字正射影像（DOM）、数字划地图（DLG），此类数据从数据类型上也属于非结构数据，数据同样定位是做备份，为将来非结构化数据分析利用做数据储备。

实时数据接入：雅砻江公司实时数据主要来自于电力生产过程中相关的传感器和设备，通过分布式控制系统（DCS）、数据采集与监视控制系统（SCADA）等底层系统进行采集，真实反映了电力生产的实时状态和历史过程。电力系统中传

感器等设备产生的数据量很大且变化速度快，具有很严格的处理时效性要求。需实现高效率、简洁、实时的数据采集和处理。

数据集成用例图如下图 3-1 所示：

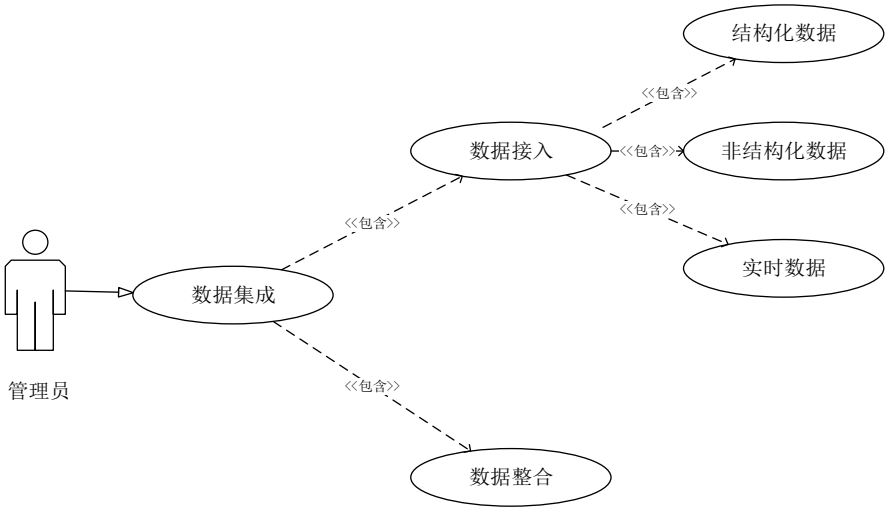


图 3-1 数据集成用例图

3.2.3 数据存储需求分析

根据对公司的数据梳理和数据集成的需求，大数据基础平台需求实现结构化数据、非结构化数据和实时数据三种数据类型的集中化、一体化的数据存储，依照统一的建模规范和标准，并可基于流域基础数据模型(SSC)实现覆盖流域全类型数据的关联。

数据存储平台需具备高扩展性，可支撑 PB 级以上规模数据在线存储，需实现海量数据的高并发读写、高效率存储、高扩展性等要求。数据存储平台易对外共享，并可实现数据间的关联流转，可同时存储各种汇总数据和指标计算数据。存储平台可支撑实现大数据的分布式计算，支撑大数据的分析和挖掘，进一步提升公司数据价值。

数据存储用例图如下图 3-2 所示：

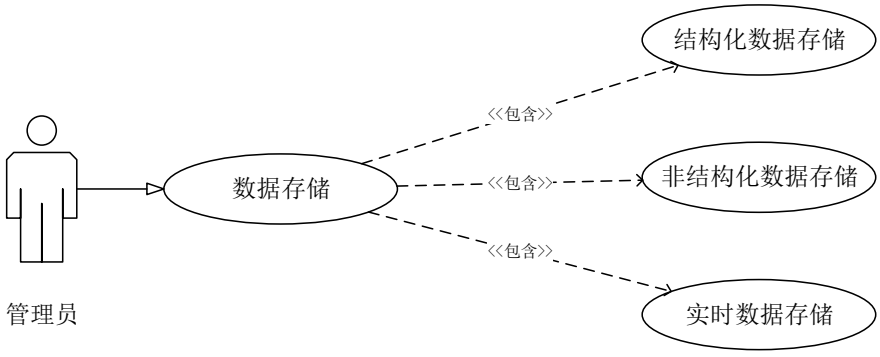


图 3-2 数据存储用例图

3.3 非功能性需求

3.3.1 性能需求

大数据基础平台快速响应类的响应时间≦10 秒（如：分析指标规则维护、接口监控管理等）；普通响应类的响应时间≦20 秒（如：指标报表展现等）；批量处理类的响应时间≦5 分钟（如：数据集成任务部署等）。

根据数据现状分析，非结构化的数据量远小于可采集的海量实时数据的数据量，大数据基础平台设计支持日 500 万条实时数据的接入。

3.3.2 可靠性

系统平台应能够连续 7×24 小时不间断工作，出现故障应能及时告警。

系统功能平台双机部署，避免单点故障，在发生系统发生单点故障时可不影响或者迅速恢复业务，单点故障时集群宕机时间<1 分钟，其余业务宕机时间<4 小时。

3.3.3 可用性

系统 7×24 小时持续可用，可在每日特定时间段内对系统进行维护。

数据存取服务要求准确，保证数据不丢失。

3.3.4 开放性和可扩展性

系统平台需具备开放性和可扩展性。

可灵活的扩展满足未来数据类型和性能发展的需求。

标准的开放接口，系统平台可将源数据和经处理后的数据共享予其他系统使用。

3.4 本章小结

本章主要从雅砻江流域水电开发有限公司的信息化现状、数据现状出发，分析了公司对数据资产的整体需求，重点对公司整体的数据集成，数据存储需求进行了分析，同时提出了系统的性能需求，作为大数据基础平台的设计基础。第六章：总结与展望。总结了大数据基础平台的建设工作及成效，并对后续大数据基础平台的完善及大数据分析工作进行了展望。

第四章 系统设计

4.1 设计目标及原则

4.1.1 设计目标

搭建雅砻江企业级大数据基础平台，形成源业务系统数据接入大数据基础平台接口建设方案、数据接入及数据存储机制，以支撑企业经营管理活动涉及到的结构化数据、非结构化数据、实时数据的抽取、转换、存储。

4.1.2 设计原则

大数据基础平台的架构设计遵循的原则：

（1）统一性：核心业务统一，核心数据模型统一，核心功能统一，业务系统接入方式统一等。

（2）先进性：模型灵活、可扩展，各项技术经济可比指标先进。

（3）经济性：综合考虑初期的建设投资与后续运行维护费用，追求大数据基础平台生命期内最佳的企业经济效益。

（4）灵活性：接口灵活，扩展灵活。适应业务类型的变化和业务规模的变化。

（5）时效性：顶层设计，创立滚动修订机制，随着业务的不断完善，需求的不断细化，不断补充和完善设计成果。

4.2 系统总体框架设计

大数据基础平台基于混合架构，是公司流域全类型数据的整合中心，系统总体框架如图 4-1 所示：

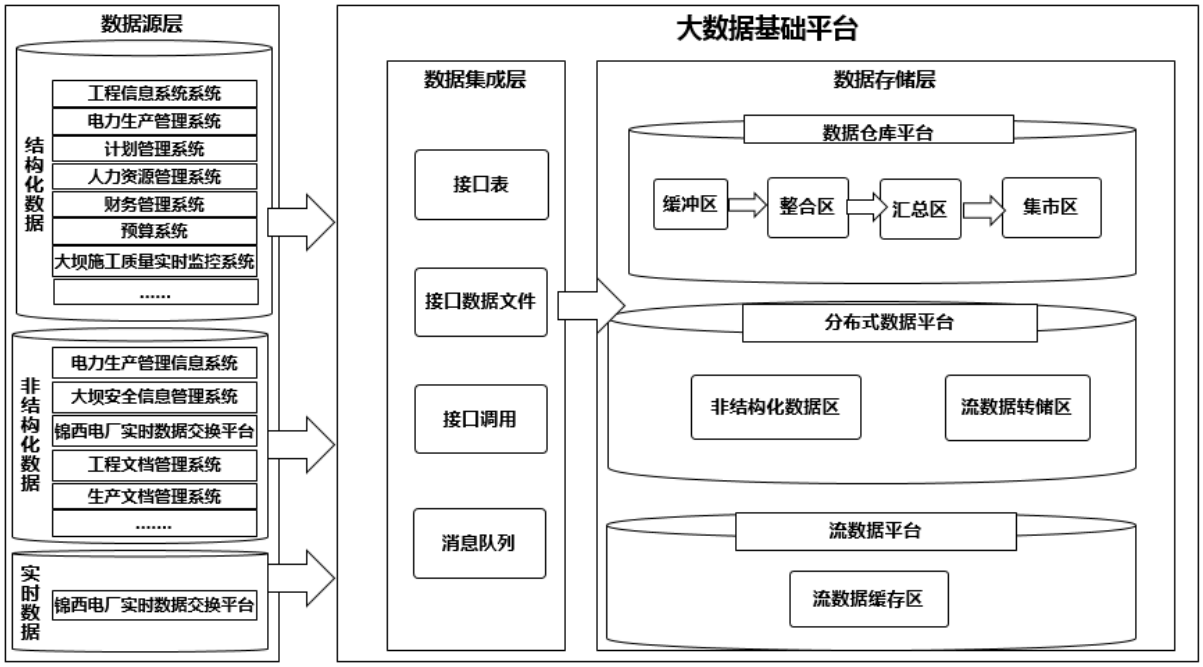


图 4-1 系统总体框架图

1、数据源层

数据源层包含了公司现有的各业务的信息系统，通过需求分析，将各信息系统的数据进行梳理后，分为结构化数据，非结构化数据和实时数据三种类型。

2、数据集成层

通过接口表、接口数据文件、接口调用、消息队列等方式，实现对数据源层中结构化数据、非结构化数据和实时数据三种类型数据的获取，根据不同数据类型，数据不同的时效性要求，分别开展作业调度，实现数据的自动收集、整理、清洗、转换，并接入到大数据基础平台的数据存储层。

3、数据存储层

数据存储层包含数据仓库平台，分布式数据平台，流数据平台。实现结构化数据、非结构化数据、实时数据分别存储到数据仓库平台、分布式数据平台、流数据平台。

4.3 数据源层系统命名设计

为加强统一管理，提升大数据基础平台数据存储的规范性，各业务系统命名规范如表 4-1 所示：

表 4-1 数据源层系统命名规范

系统标识	系统名称	系统英文简称
S01	工程管理系统（现场）	WPMS
S02	电力生产管理信息系统	EPPMS
S03	人力资源系统	HRMS
S04	财务管理系统	FMS
S05	预算系统	BMS
S06	大坝安全信息管理系统	DSIMS
S07	大坝施工质量实时监控系统	DCQRTMS
S08	集控中心数据交换平台	HPDXP
S09	锦西电厂实时数据交换平台	HPRTDMS
S10	工程文档管理系统	PDMS
S11	生产文档管理系统	EPPDMS
S12	档案管理系统	AMS
S13	统一采购平台	BTMS
S14	流域公共安全信息管理与决策支持系统	WPSMS
S15	流域征地移民信息管理与决策支持系统	WLEMS
S16	流域环保水保信息管理与决策支持系统	WECMS
S17	电力营销系统	EEMS
S18	锦屏一级设备状态评价系统	ESEMS

4.4 数据集成层设计

基于源端业务应用系统的数据现状，数据集成的方法将具备多样性[33]。针对大数据基础平台与外围系统数据集成需求，从场景描述、技术方式、数据特征、触发机制、处理步骤等维度总结如下几种数据集成场景：

4.4.1 结构化数据集成设计

4.4.1.1 大数据基础平台提供接口表方式

对数据量一般，时效性要求不高的结构化数据，通过接口表方式，由源系统将结构化数据批量推送到大数据基础平台数据仓库的缓冲区中，如图 4-2 所示：

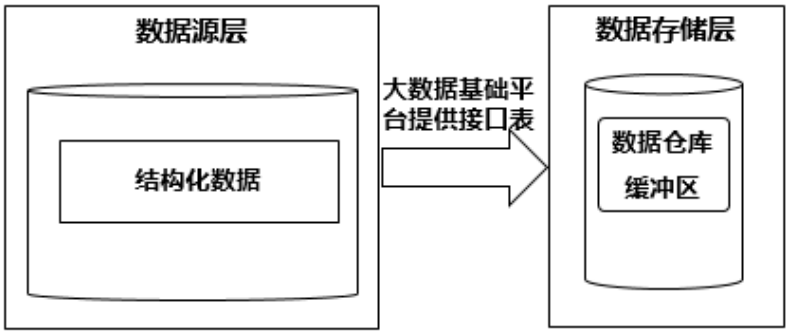


图 4-2 大数据基础平台提供接口表方式框架图

大数据基础平台提供接口表方式集成结构化数据的具体处理流程如图 4-3 如示：

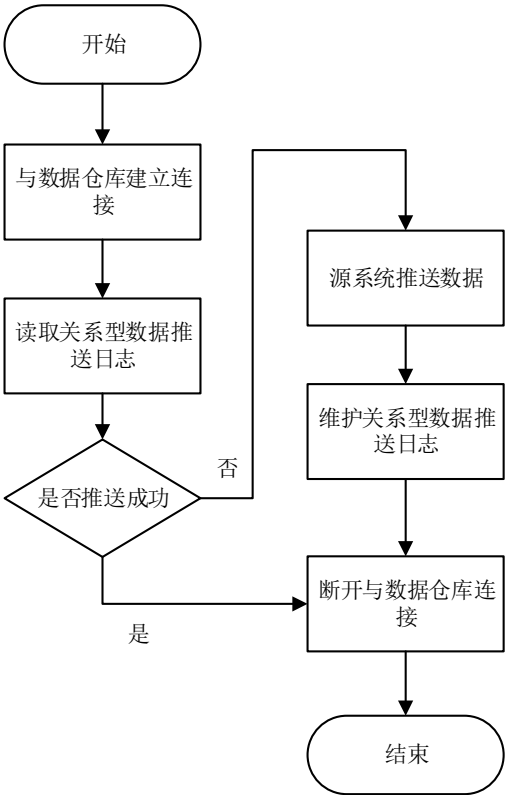


图 4-3 大数据基础平台提供接口表方式流程图

处理步骤：

1.大数据基础平台在数据仓库缓冲区依照原系统数据表结构，采用贴源设计为源系统建立相关的接口表，并为源系统提供具有写接口表权限的数据库用户；

2.源系统通过 ETL 方式按照约定的数据传输时间定时连接到数据仓库；

3.源系统读取数据仓库中的关系型数据推送日志，判断数据的更新周期内是否有数据推送成功，若周期内推送成功则断开数据仓库连接；若周期内无推送记录或者推送失败，则源系统推送相关表的全量或增量数据到数据仓库缓冲区的接口表中；

4.推送结束后维护关系型数据推送日志，标记推送成功或者失败信息。

4.4.1.2 源系统提供接口表方式

对数据量一般，时效性需求较低的结构化数据，通过源系统接口表方式，源系统按照双方约定的数据采集范围、数据提供频率将数据同步到源系统接口表中，大数据基础平台从源业务系统接口表中获取数据，如图 4-4 所示：

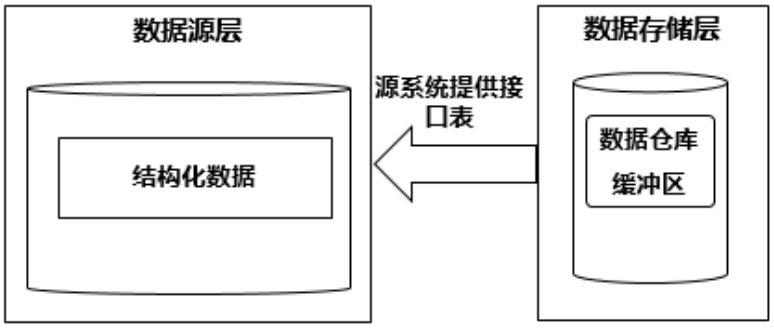


图 4-4 源系统提供接口表方式框架图

源系统提供接口表方式集成结构化数据的具体处理流程如图 4-5 如示：

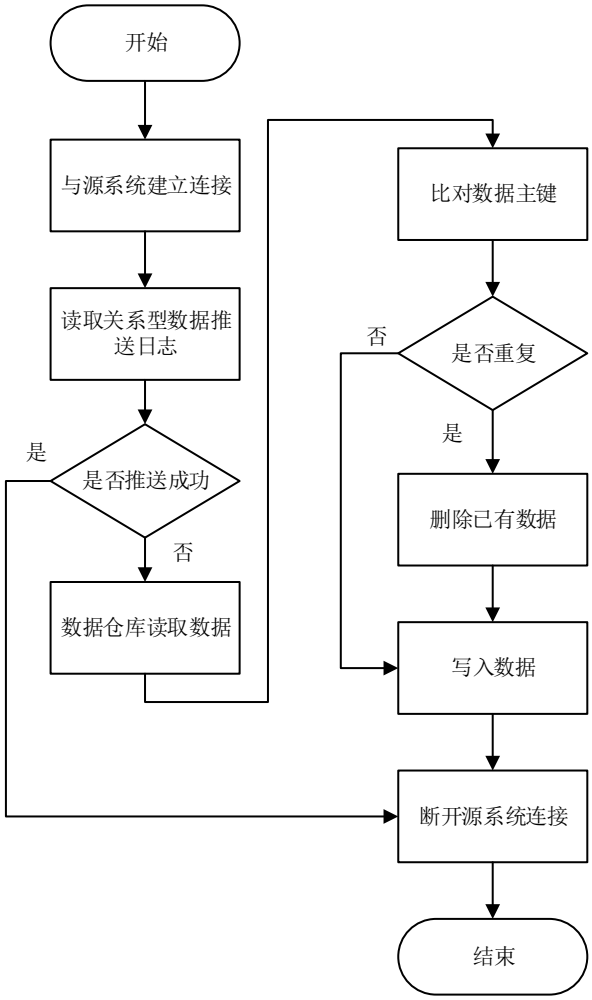


图 4-5 源系统提供接口表方式流程图

处理步骤：

1.源系统为大数据基础平台新建数据库只读用户；

2.大数据基础平台按规定周期时间执行任务，通过 ETL 方式连接到源系统数据库；

3.大数据基础平台检查关系型数据推送日志，判断更新周期时间内是否有数据抽取成功，若周期内抽取成功则断开与源系统数据库的连接；若无抽取操作或者抽取失败，则通过 ETL 方式抽取源系统数据库相关表的全量或增量数据

4.大数据基础平台对取得的数据根据数据表的主键，进行比对。删除与新数据具有相同主键的旧数据，写入抽取的新数据，存储到数据仓库的缓冲区中，保证数据一致性。

5.抽取结束后维护关系型数据推送日志，标记抽取成功或者失败信息。

4.4.1.3 ETL 作业调度设计

ETL 作业调度采用 Linux 操作系统自带的调度程序 Crontab，为了保证 ETL 作业的有效运行，避免大量作业同时启动导致服务器资源耗尽，亦或是作业中断导致部分数据表未处理的情况，设计如下规则：

1.每个作业启动间隔为 2 分钟以上，避免作业同时启动，确保作业稳定运行；

2.为避免因作业中断导致部分数据表未处理的情况，每个作业的重复间隔为 1 小时，通过 ETL 作业逻辑控制需要处理的数据表，防止作业重复运行导致数据表重复处理；

3.由于作业重复间隔时间短，存在作业启动时，上次作业进程未结束的情况，因此在作业启动时，需要判断当前是否存在相同的作业进程，若存在则不启动本次作业。

具体流程如图 4-6 所示：

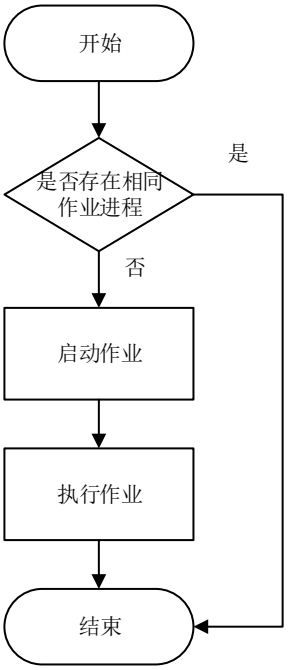


图 4-6 ETL 作业调度流程图

4.4.2 非结构化数据集成设计

4.4.2.1 数据文件方式

对数据量较大，时效性需求较低的非结构化数据，通过 FTP 文件传输的方式，将非结构化数据集成至大数据基础平台文件缓冲区，如图 4-7 所示：

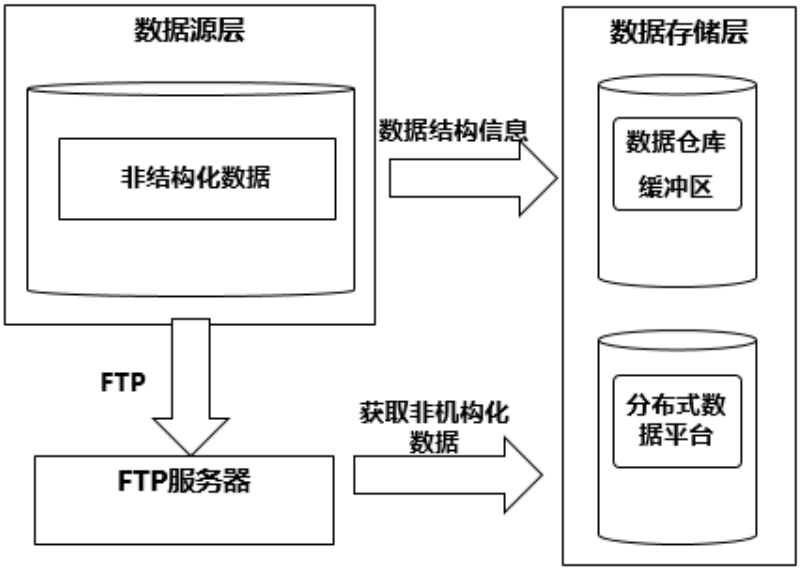


图 4-7 数据文件方式框架图

数据文件方式集成非结构化数据的具体处理流程如图 4-8 如示：

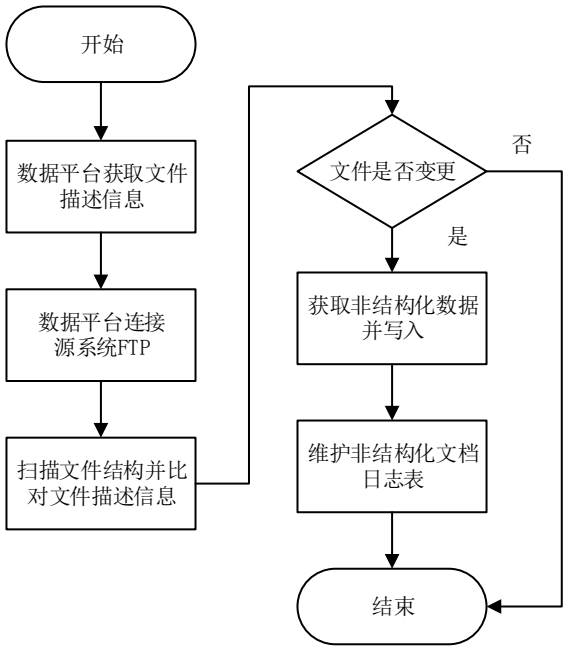


图 4-8 数据文件方式流程图

处理步骤：

1.设立 FTP 服务器，将相关权限分配给大数据基础平台和源系统。

2.源系统在本地维护附件文件名，文件类型，路径，大小，创建时间等文件描述信息记录。大数据基础平台通过 ETL 方式取得文件描述信息记录，存放至数据仓库缓冲区的非结构化文档日志表中。

3.源系统负责定时将相关非结构化数据及对应的描述文件按约定的存放规划、命名规则，通过 FTP 的方式上传至 FTP 服务器上。

4.大数据基础平台访问 FTP 服务器，根据数据仓库缓冲区中的非结构化文档日志表扫描 FTP 路径树，与数据仓库缓冲区中的非结构化文档日志表中的文件信息进行比对，如果文件名、格式、文件大小、创建时间等相关信息未发生变动，不做处理。如果没有记录或者记录有变化，通过二进制的方式读取大数据基础平台 FTP 中的文件，并将该二进制数据写入到大数据基础平台分布式大数据平台 HBase 库中。

5.同时将非结构化文件的文件名、文件格式、文件大小、创建时间以及写入 HBase 库中非结构化文件唯一标识（即 HBase 行键）写入到非结构化文档日志表中。

4.4.2.2 接口调用

对数据量较大，时效性需求较低的非结构化数据，通过调用源系统接口的方式，从源系统主动获取非结构化数据文件至大数据基础平台分布式数据平台，如图 4-9 所示：

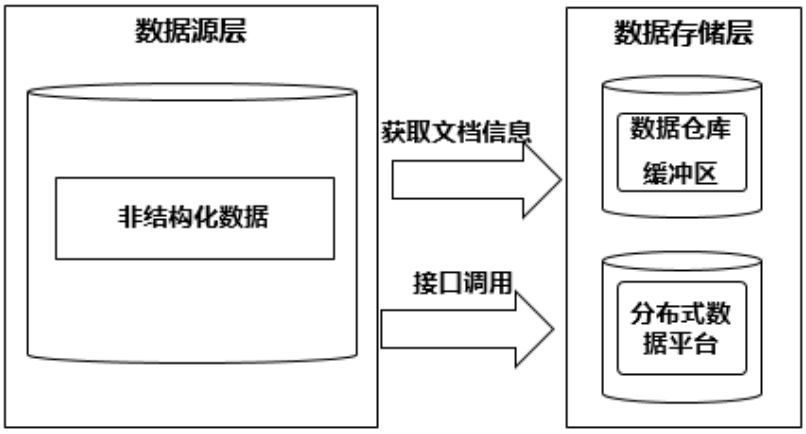


图 4-9 接口调用方式框架图

接口调用方式集成非结构化数据的具体处理流程如图 4-10 如示：

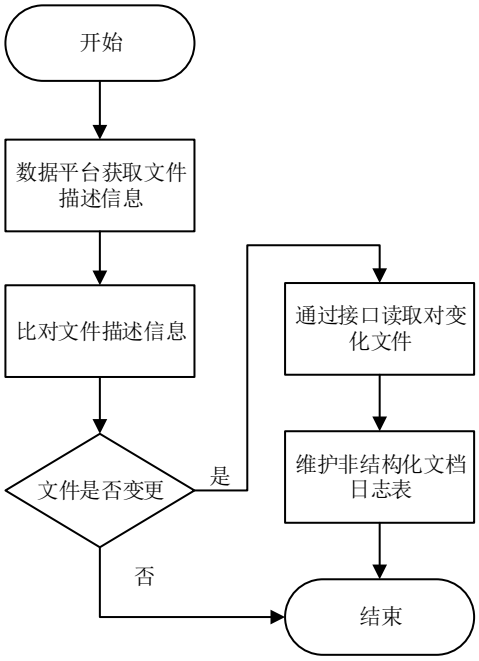


图 4-10 接口调用方式流程图

处理步骤：

1.源业务系统提供非结构化数据访问的相关权限信息（IP、端口）；

2.源系统维护一个记录，包含文件树信息，文件 ID，文件格式，创建时间等信息。大数据基础平台通过 ETL 方式获取，存放至数据仓库缓冲区的对应的非结构化文档日志表。

3.通过非结构化文档日志表中的信息判断文件是否有变化，如果有就通过源系统提供的接口依照文件 ID 去读取文件，实现将源业务系统增量的非结构化数据存放到分布式数据平台中。

4.同时将非结构化文件的文件名、文件格式、文件大小、创建时间以及写入 HBase 库中非结构化文件唯一标识（即 HBase 行键）写入到数据仓库缓冲区对应的非结构化文档日志表。

4.4.3 实时数据集成设计

现公司的实时数据为电厂监控系统系统产生的实时监控数据，其数据量大，时效性需求高，大数据基础平台实时数据接集成采用消息队列组件（Kafka）+流计算组件（Storm）的方式，实现公司生产实时数据的高效集成，如图 4-11 所示：

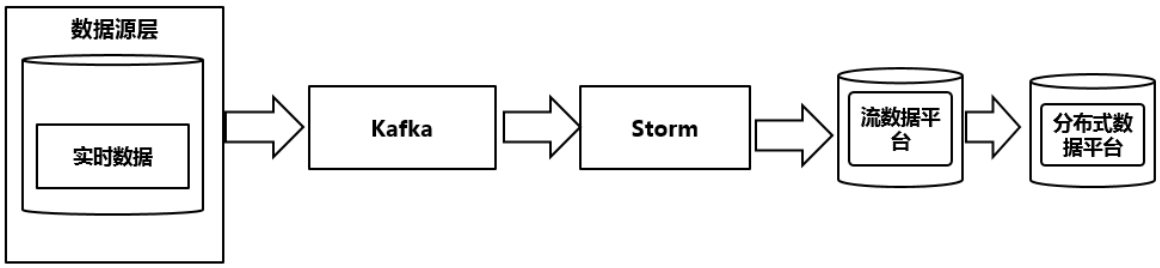


图 4-11 实时数据集成框架图

实时数据集成的具体处理流程如图 4-12 如示：

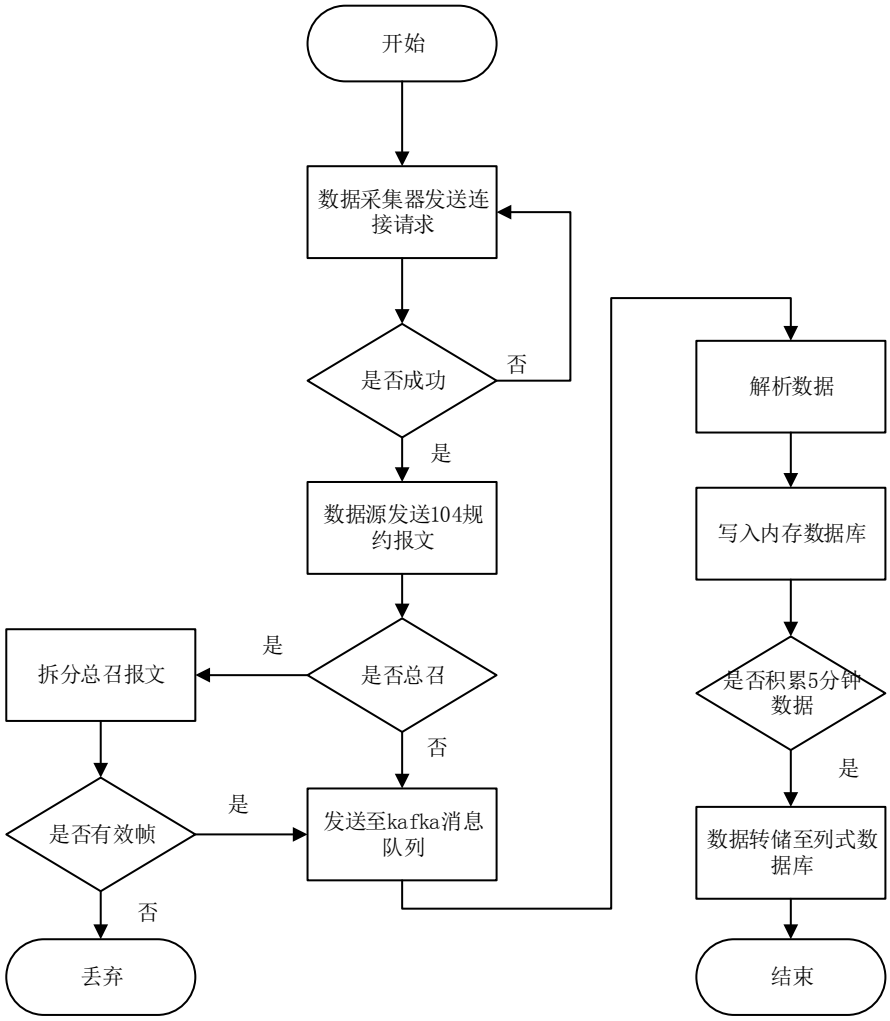


图 4-12 实时数据集成流程图

处理步骤：

1.大数据基础平台向实时数据服务端提供接口发送连接建立请求，连接建立成功后，大数据基础平台周期性发送测试帧，监测链路是否断开，如果检测到链路断开，则自动重新建立连接。

2.根据规约，连接建立后，服务端开始向大数据基础平台发送变化的实时数据。

3.大数据基础平台同时会周期性发送一次总召报文到服务端，服务端收到总召报文后，会将所有监控测点的实时数据全部发送一遍。

4.大数据基础平台 Socket 接收到数据后，将根据规约，按帧拆分接收到的数据包，判断是否为数据帧，丢弃控制帧和测试帧。

5.将数据帧发送至 Kafka 中消息队列中，Kafka 集群会实现数据的持久化工作。

6.通过 Storm 流处理技术，从 Kafka 获取消息，照约定的格式定义对原始报文解析。

7.解析结果在 Redis 内存数据库中进行存储。

8.按定义时间周期性连接 Redis，读取时间片的数据，存储至 HBase 中，然后清除 Redis 中对应的数据。

4.5 数据存储层设计

基于雅砻江业务系统数据类型和应用需求，设计搭建混合架构的大数据存储层，将数据存储区域划分为存储结构化数据的数据仓库平台，存储非结构化数据的分布式数据平台，以及存储实时数据的流数据平台[34-35]，如图 4-13 所示：

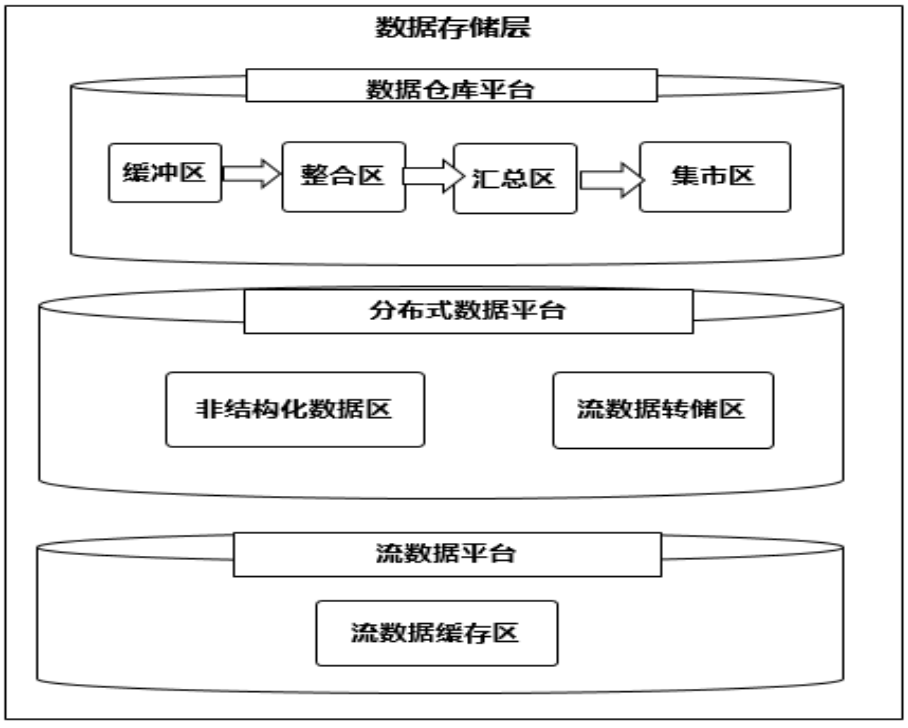


图 4-13 数据存储层框架图

4.5.1 数据仓库平台设计

4.5.1.1 总体设计

数据仓库建设，需制定统一的建模规范和标准，并结合业务需求，专门针对企业数据整合和数据历史存储需求而组织的集中化、一体化的数据存储。这些数据按照业务主题域进行分类和存储，存储的在线周期一般要求比较长，主要包括低级别、细粒度数据，同时可以根据数据分析需求通过 ETL 技术进行汇总和指标计算，并按照一定频率定期更新，以支撑业务场景监测分析以及数据挖掘。

数据仓库平台采用国产关系型数据库 GBase 构架，进行分区设计，分为了缓冲区、整合区、汇总区和集市区，如图 4-14 所示：

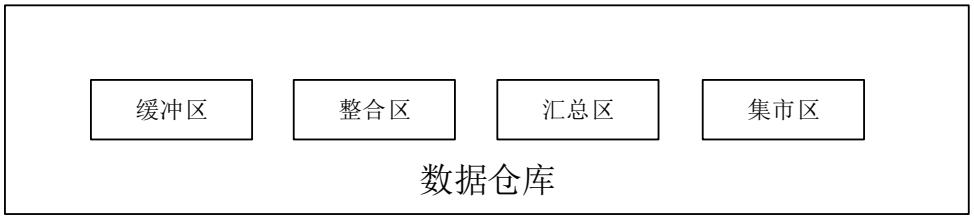


图 4-14 数据仓库框架图

缓冲区

缓冲区 Origin Data Model，简称 ODM，大数据基础平台从源系统采集的结构化数据存储于此，设计如下：

1.缓冲区独立的存储各类原始数据

2.缓冲区采集的结构化数据与源系统数据的结构一致。

3.增加技术列设计，满足数据集成时的多份保存，追踪数据版本变化等需求。

4.缓冲区担负数据集成时的性能需求。

整合区

整合区 Foundation Data Model，简称 FDM，整合后的数据存储于此，是大数据基础平台的核心数据层，设计如下：

1.对数据存储的格式进行统一规范，使数据逐步统一，整合区的规范也是对企业后续信息系统建设时候数据规范形式的参考。

2.对数据按照规划进行整合后存储，是 SSC 企业数据模型的落地区。

汇总区

汇总区 Aggregation Data Model，简称 ADM，轻度汇总后的数据存储于此，加速集市区输出报表，设计如下：

1.根据各类报表的需求，对整合层的数据进行对应维度（如时间，电站）的汇

总转换。

2.支持多层级汇总。

数据集市

集市区 Mart Data Model，简称 MDM，从专业业务需求角度出发，按需构建数据集市区，集市区可通过对数据结构的索引针对用户的访问和数据输出实施优化。大数据基础平台主要以下三类集市：

1.应用集市：支撑某个应用系统的数据分析需求，分析型应用不单独存储数据，需存放在数据平台应用集市中。如果应用数据较少，可保存在部门集市或电站分集市中，不单独建立应用集市；

2.部门集市：支撑公司各部门内部，以及电站对应部门考核的数据分析需求，可以采用物理方式或逻辑方式；

3.电站集市：支撑各电站内部的数据分析需求，采用物理方式，只存放不同电站个性化应用分析相关的数据。

4.5.1.2 数据库设计

根据雅砻江企业级大数据基础平台总体架构及数据架构的设计，以及企业级大数据基础平台数据仓库平台产品 GBase 8T 产品的特点，雅砻江大数据基础平台数据仓库平台从数据库层面分为以下四个数据库：缓冲区数据库、整合区数据库、汇总区数据库、数据集市区数据库，如表 4-2 所示：

表 4-2 数据仓库数据库设计

名称	数据库
缓冲区数据库	Odmsitdb
整合区数据库	Fdmsitdb
汇总区数据库	Admsitdb
数据集市区数据库	Mdmsitdb

4.5.1.3 DBSpace 设计

GBase 数据库中，DBSpace 是基于 Chunk 组合的数据空间，Chunk 实际时磁盘上一块连续的物理空间，以页为基本单位。DBSpace 规划时考虑预留后续存储空间需求，保持分配 Chunk 的物理连续性，可减少数据读写时间，优化性能。根据数据需求分析,大数据基础平台数据仓库平台 DBSpace 设计如表 4-3 所示：

表 4-3 DBSpace 设计

Dbspace	Chunk	Size(GB)	备注
rootdbs	rootdbs	10	根数据空间
logdbs	logchk1	150	逻辑日志空间
	logchk2	150	
phydbs	phychk1	50	物理日志空间
tmpdbs1	tmpchk1	50	临时文件空间
tmpdbs2	tmpchk2	50	临时文件空间
tmpdbs3	tmpchk3	50	临时文件空间
tmpdbs4	tmpchk4	50	临时文件空间
sdstmpdbs_gbase1	sdstmpdbs_gbase1	50	跨库临时表空间
sdstmpdbs_gbase2	sdstmpdbs_gbase2	50	跨库临时表空间
sds_alt_comm	sds_alt_comm	1	交互空间
sbdbs	sbchk1	100	大字段存储空间
	sbchk2	100	
	sbchk3	100	
	sbchk4	100	
	sbchk5	100	
odmprdspace01	odmprdchk01	200	缓冲区空间
	odmprdchk02	200	
	odmprdchk03	200	
	odmprdchk04	200	
	odmprdchk05	200	
	odmprdchk06	200	
	odmprdchk07	200	

Dbspace	Chunk	Size(GB)	备注
	odmprdchk08	200	
	odmprdchk09	200	
	odmprdchk10	200	
fdmprdspace01	fdmprdchk01	200	整合区空间
	fdmprdchk02	200	
	fdmprdchk03	200	
	fdmprdchk04	200	
	fdmprdchk05	200	
	fdmprdchk06	200	
	fdmprdchk07	200	
	fdmprdchk08	200	
	fdmprdchk09	200	
	fdmprdchk10	200	
admprdspace01	admprdchk01	200	汇总区空间
	admprdchk02	200	
	admprdchk03	200	
	admprdchk04	200	
	admprdchk05	200	
	admprdchk06	200	
	admprdchk07	200	
	admprdchk08	200	
	admprdchk09	200	
	admprdchk10	200	

4.5.1.4 用户设计

数据库用户按用户类型一般分为 DBA 用户、属主（OWNER）用户、访问用户。并基于安全性考虑，设置公共的只读用户。

根据雅砻江大数据基础平台数据仓库平台的功能，应考虑按数据平台数据逻辑层级及物理分布划分不同用户，用户设计如表 4-4 所示：

表 4-4 数据仓库用户设计

用户名	角色	权限
UDBA	DBA 用户	DBA 权限，作为管理员账户使用。
UADM	汇总区相关表的属主	对 ADM 表的 DDL 权限（Create、Drop、Alert 等）
UFDM	整合区相关表的属主	对 FDM 表的 DDL 权限（Create、Drop、Alert 等）
UODM	缓冲区相关表的属主	对 ODM 表的 DDL 权限（Create、Drop、Alert 等）
UMDM	数据集市区相关表的属主	对 MDM 表的 DDL 权限（Create、Drop、Alert 等）
UADMOPR	汇总区访问用户	对 ADM 表数据的 DML 权限（Insert、Update、Delete、Select）
UFDMOPR	整合区访问用户	对 FDM 表数据的 DML 权限（Insert、Update、Delete、Select）
UODMOPR	缓冲区访问用户	对 ODM 表数据的 DML 权限（Insert、Update、Delete、Select）
UMDMOPR	数据集市区访问用户	对 MDM 表数据的 DML 权限（Insert、Update、Delete、Select）
URO	只读用户	对上述表数据的只读权限（Select）

4.5.1.5 业务数据表设计

缓冲区数据库表采用贴源设计，与需集成的源系统对应数据表结构完全一致。

整合区数据库表根据 SSC 企业数据模型主题，对数据进行整合建表。

汇总区数据库表基于整合区的主题域，根据报表需求，根据时间、部门等维度值进行轻度汇总，加速集市区输出报表。

集市区是按需设计出报表的地方，例如公司水情报表，以电站、日期为维度，水位、流量为值。

4.5.1.6 配置信息表和日志信息表设计

数据仓库采用分区设计，采用 ETL 方式实现数据集成，数据会由源系统集成至缓冲区，由缓冲区集成至整合区，由整合区集成至汇总区，由汇总区集成至集市区。平台设计了作业配置表（kjb_xxx_to_yyy_cfg）、依赖关系表（dep_xxx_to_yyy_cfg）、等配置信息表，以及作业日志表（etl_kjb_log）、数据表日志表（etl_table_log）、错误记录表（log_xxx_to_yyy_errcnt）等日志信息表，用于控制数据流转的作业运行。同时还设计了非结构化数据文档信息表（系统编码_FILE_SIZE_INC），记录集成的非结构化数据信息。下面将具体介绍各项表的设计：

1.作业配置表

作业配置表记录 ETL 作业中涉及的转换名称、转换路径、目标表名称。用于指明 ETL 资料库中的具体的转换，转换内保存了具体的逻辑映射关系。

作业配置表针对每个源系统单独建立一张表，命名 kjb_xxx_to_yyy_cfg，xxx 为源系统名称或者数据仓库分区名称，yyy 为数据仓库分区名称。例如电力生产系统到缓冲区的ETL作业配置表名为kjb_eppms_to_odm_cfg,缓冲区到整合区的ETL作业配置表名为 kjb_odm_to_fdm_cfg.作业配置表结构如表 4-5 所示：

表 4-5 作业配置表

字段	数据类型	是否为空	备注
ktr_name	varchar(255)	not null	转换名称
ktr_path	varchar(255)	not null	转换路径
target_table	varchar(50)	not null	目标表名称

2.依赖关系表

依赖关系表记录 ETL 作业的目标表名称、来源表名称、是否 ETL 集成。用于记录表的整合关系，同时目标表名称和来源表名称分别与数据表日志表（etl_table_log）中 table_name 字段关联，若所有的来源表的状态均为“Finish”并且

结束时间均大于目标表的结束时间，则可确认此数据表依赖的来源表已完成更新，该数据表具备更新的条件。

依赖关系表对每个源单独一张表，命名 dep_xxx_to_yyy_cfg，xxx 为源系统名称或者数据仓库分区名称，yyy 为数据仓库分区名称。例如电力生产系统到缓冲区的 ETL 依赖关系表名为 dep_eppms_to_odm_cfg，缓冲区到整合区的 ETL 依赖关系表名为 dep_odm_to_fdm_cfg。依赖关系表结构如表 4-6 所示：

表 4-6 依赖关系表

字段	数据类型	是否为空	备注
target_table	varchar(50)	not null	目标表名称
source_table	varchar(50)	not null	来源表名称
etl_flag	varchar(50)	not null	ETL 标识

3.作业日志表

作业日志表记录 ETL 作业的作业名称、作业开始时间、作业结束时间和作业执行状态。用于实现监控 ETL 作业的执行状态。作业日志表全局唯一，其表结构如表 4-7 所示：

表 4-7 作业日志表

字段	数据类型	是否为空	备注
kjb_name	varchar(50)	not null	作业名称
begin_dttm	datetime		开始时间
end_dttm	datetime		结束时间
Status	varchar(20)		状态

4.数据表日志表

数据表日志表记录数据表名称、数据表更新开始时间、数据表更新结束时间、数据表状态、数据表更新行数等信息。用于实现监控数据表更新的执行状态。数据表日志表全局唯一，其表结构如表 4-8 所示：

表 4-8 数据表日志表

字段	数据类型	是否为空	备注
table_name	varchar(50)	not null	数据表名称
begin_dttm	datetime		开始时间
end_dttm	datetime		结束时间
Status	varchar(20)		状态
row_count	integer(9)		行数

5.错误记录表

错误记录表记录转换执行失败次数，用于作业失败重试控制。若转换执行失败，将该转换对应的数据表名称添加到错误记录表中，并统计错误记录表中该表名称的数量，若数量小于等于配置的重试次数，则重新执行该转换，若数量大于配置的重试次数，则停止执行该转换，并将该数据表的更新状态标记为“Failed”，并执行下一个转换。错误记录表会在作业初始化时清空，以保证每次作业执行时，各个数据表的错误记录数量都是从零开始。

错误记录表对每个源单独一张表，命名 log_xxx_to_yyy_cfg，xxx 为源系统名称或者数据仓库分区名称，yyy 为数据仓库分区名称。例如电力生产系统到缓冲区的 ETL 错误记录表为 log_eppms_to_odm_cfg，缓冲区到整合区的 ETL 错误记录表名为 log_odm_to_fdm_cfg。错误记录表结构如表 4-9 所示：

表 4-9 错误记录表

字段	数据类型	是否为空	备注
target_name	varchar(50)	not null	数据表名称

6.非结构化文档日志表

非结构化文档日志表记录非结构化数据集成的文件大小，文件类型，创建时间，UNSTRUCT_ID 等信息，用于记录数据仓库缓冲区已集成的非结构化数据信息，与待集成数据信息进行比对。每一个源系统维护一个非结构化文档日志表，设计命名为系统编码_FILE_SIZE_INC。其中 UNSTRUCT_ID 由系统编码+源系统主键组成。非结构化文档日志表结构如表 4-10 所示：

表 4-10 非结构化文档日志表

字段	数据类型	是否为空	备注
UNSTRUCT_ID	varchar(255)	not null	非结构化文档编码
FILE_SIZE	integer（9）		文件大小
FILE_TYPE	varchar（10）		文件类型
CREATION_DATE	datetime		创建时间
Md5_UNSTRUCT_ID	varchar（32）		非结构化存储行键

4.5.1.7 表分片设计

分片表可以支持除了在分片主键上进行创建本地索引外，还可以创建多个其他非分片键字段的本地索引，而在对该表进行分片管理时（删除一个分片，增加一个分片）不影响表的使用，索引不需要重建。

对于数据量大于 2G 或者按时间归档的历史表可以考虑建立表分片，分区一般使用表达式分区策略创建。

4.5.1.8 技术列设计

1.技术主键

因为缓冲区存储多个源系统的多份数据，源系统的主键存在重复的可能，源系统的主键不能在大数据基础平台作为主键使用。本系统统一对每个表设计了单列的技术主键，其仅仅作为数据库每张表的主键。命名统一采用 xxID，数据类型根据数据量使用 BIGINT 或 INTEGER.。

2.审计追踪列

作为企业的大数据基础平台，所有数据集中于此，为便于追溯每条数据的变更痕迹，本系统针对每个数据表设计了审计追踪列，审计追踪列包括的列如表 4-11 所示：

表 4-11 审计追踪列

列名	**说明**	**数据类型**	**备注**
DEL_FLAG	更新标志	VARCHAR(2)	**即逻辑删除标志**
ETL_CREATE_DTTM	数据抽取创建时间	DATETIME YEAR TO FRACTION	**记录插入的系统时间**
ETL_UPDATE_DTTM	数据抽取更新时间	DATETIME YEAR TO FRACTION	记录最后修改系统时间

4.5.1.9 索引设计

针对大数据基础平台数据的设计使用情况，索引的建立可加快查询的响应速度，提升平台的处理性能，但会占用更多的存储空间。从企业的业务系统主键的角度，对查询频繁的字段梳理了以下关键的索引，如表 4-12 所示：

表 4-12 索引设计

主题代码	表名称	索引字段
T01	T01_SSC	SSC_CD
T02	T02_DESIGN_QUANTITIES_LIST	DESIGN_DETAIL_CD
T02	T02_BUILD_DRAWING	DRAWING_CD
T02	T02_BUILD_DRAWING_ITEM	BUILD_DRAWING_ITEM_CD
T02	T02_CTS_CONTRACT	CONTRACT_CD
T03	T03_EQUIP_ASSET	KKS_CD
T03	T03_KKS	KKS_CD
T03	T03_KKS_HIERARCHY	KKS_CD
T03	T03_EQUIP_SPAREPART	MAT_CD
下略		

4.5.2 分布式数据平台设计

4.5.2.1 总体设计

分布式数据平台基于 Hadoop 构建，利用 HBase 实现数据存储区，分为非结构化数据区，流数据转储区，如图 4-15 所示：

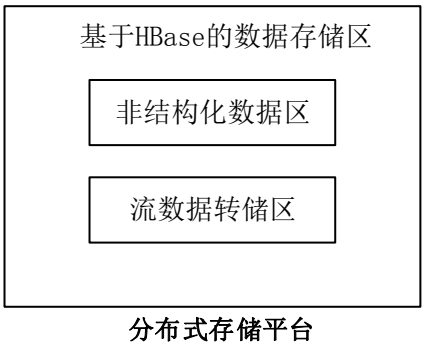


图 4-15 分布式数据平台框架图

4.5.2.2 非结构化数据区设计

4.5.2.2.1 存储内容

根据前期需求，接入到数据中心分布式存储平台的非结构数据主要涉及生产文档管理系统（FILENET)、工程文档管理系统（UCM）、大坝安全信息管理系统、档案管理系统、流域三维可视化与信息集成展示与会商平台、流域公共安全信息管理与决策支持系统、流域征地移民信息管理与决策支持系统、流域环保水保信息管理与决策支持系统。大数据基础平台通过流文件的形式读取文件内容并存储到 HBase 中，同时建立其与结构化数据的关联并存放在数据仓库相关表。

数据内容包括设计文档、检测、测试、巡视等信息文档、质量安全文档、工程设计图纸和文档、报告、巡检文档、图片、档案、归档文件等，同时也包括空间地理信息和工程三维模型信息。

4.5.2.2.2 表设计

表名

按照文档来源来进行分表的原则，每个文档来源单独的建立一张表来保存该来源系统的非结构化文件。表的命名原则可以按来源系统名或者拼音缩写来进行命名。

预分区

目前综合系统性能和信息保存现状考虑，由于无法准确的预测出文档的命名规则和分区边界，目前不考虑进行预分区。后续根据实际的数据情况进行手工分区维护。

行键

非结构化数据写入 HBASE 时，带着 UNSTRUCT_ID，对 UNSTRUCT_ID 进行 MD5，得到的值作为行键。

列族

定义了两个列族 Attr 和 Content，其中列族 Attr 用来保存文件的属性信息，如文件名，文件类型，创建时间，创建者，文件大小等。列族 Content 用来保存文件的二进制内容。

版本数控制

文件的最大版本数设置为 1。

保存时间

而目前公司对文件的保存时间要求永久保存，保存时间设置为

Integer.MaxValue 即可。

具体表结构设计如如表 4-13 所示：

表 4-13 非结构化数据区表结构

行键	列族	列	备注	数据类型
MD5(UNSTRUCT_ID)	Attr	UNSTRUCT_ID	系统编码_源系统主键	String
		FILE_SIZE	文件大小	String
		FILE_TYPE	文件类型	String
		CREATION_DATE	创建日期	String
	Content	CONTENT	文档内容	String

4.5.2.2.3 非结构化数据与数据仓库关联设计

当非结构化数据在分布式平台中保存成功后，会把文档的行键返回给数据仓库并保持在相应的物理表中。数据仓库可以根据物理表中的行键和文档来源系统确定唯一的文档路径。访问 HBASE 数据时，UNSTRUCT_ID 可关联到数据仓库的信息，访问数据仓库信息时，对 UNSTRUCT_ID 做 MD5，可关联到 HBASE，从而建立和分布式数据平台非结构化存储区的关联，如图 4-16 所示：

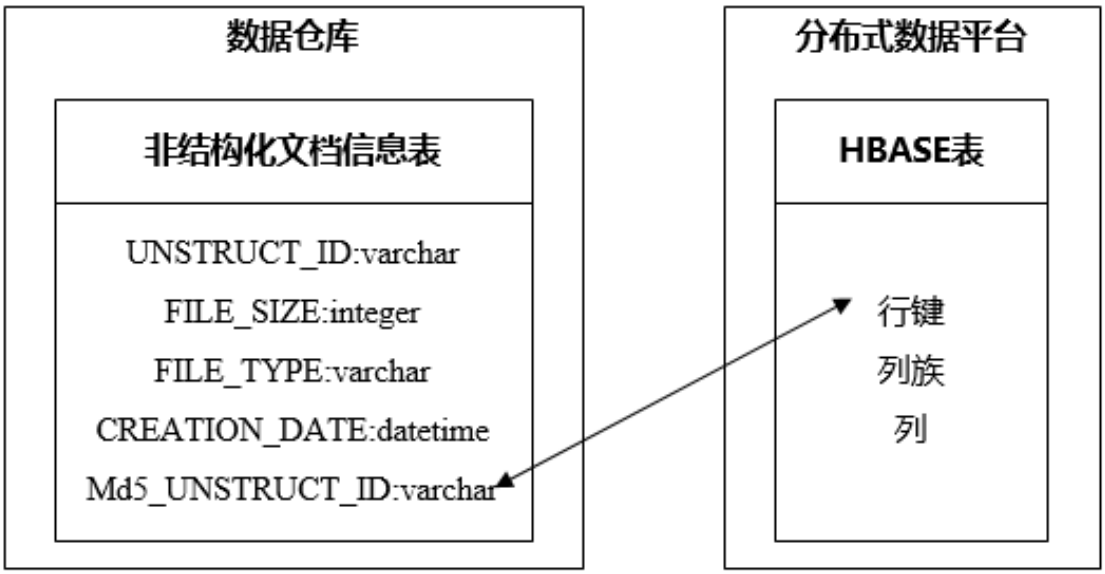


图 4-16 非结构化数据与数据仓库关联

4.5.2.3 流数据转储区模型设计

4.5.2.3.1 存储内容

实时消息数据的产生可能达到最高每秒几万甚至几十万记录的速度，流计算处理模块需要快速的处理消息，并将处理完成后的消息数据进行持久化保存。实时消息数据经 Storm 解析后先写入高性能的 Redis 内存数据库中，但由于内存限制，Redis 不适用海量数据的永久保存，因此需要将 Redis 中的数据周期转储到 Hbase 的流数据转储区中。

4.5.2.3.2 表设计

表名

业务系统中一般按照电站来进行统计和查询，不需要进行跨站比较，出于数据量控制和分区隔离的考虑，我们按照每个电站一张表的原则来进行建表。以电站名+流数据来源系统名称+链路 ID（对方的 IP 地址和端口）组成字符串作为 HBase 表名。如锦西计算机监控系统数据流数据转储表命名为“JinXi_ZXJC_链路 ID”。

预分区

目前综合系统性能和信息保存现状考虑，由于无法准确的预测数据量和增长情况，目前不考虑进行预分区。建议由管理员根据实际的数据情况进行手工分区。

列族

在每个 Rowkey 上会只定义一个列族。

版本数控制

根据实际业务要求，我们每条数据只需保留最近的一个版本即可。最大版本数设置为 1

保存时间

由于 HBase 有数据失效性的设置，当文件的存活时间超过设置的保存时间后，HBase 会在做合并的时候删除掉该记录。而目前公司对监控数据的保存时间要求永久保存，保存时间设置为 Integer.MaxValue 即可。

行键

经过前期的需求分析，前期有两个链路发生数据到我们数据接收平台，并且两个链路内可能测点的 ID 不是唯一编号，即链路一内和链路二内同一个 ID 可能代表的是不同测点，考虑到兼顾系统的查询性能和避免数据热点的要求，我们以 Redis 中的 Key 为主键，即”电站|测点|时间”。

列

列中保存的值为该五分钟内测点上送的测量时间和测值拼接而成的 JSON 字符串。如{2016-10-22 15:12:24：21.42，2016-10-22 15:13:09：24.32}。，每个值就是一个 CELL。具体时间作为每个 CELL 的名字，精确到毫秒，CELL 里面就是对应的测点值。

在系统正常运行状态下，数据量多数为稳态数据，考虑到存储大量的同样的值是无意义的，考虑在五分钟内只保留变化的数据，即只有当数据发生变化的时候才保持一条记录，如果两次数据相比较没有变化，不需要保存。

具体表结构设计如表 4-14 所示：

表 4-14 流数据转储区表结构

行键	列族	列	备注
电站|测点|时间	Data	Data,Data…	按时间片集合的测值

4.5.3 流数据平台设计

4.5.3.1 总体设计

流数据平台基于“Kafka+Storm+Redis”技术架构进行实现，流数据平台整体架构如图 4-17 所示：

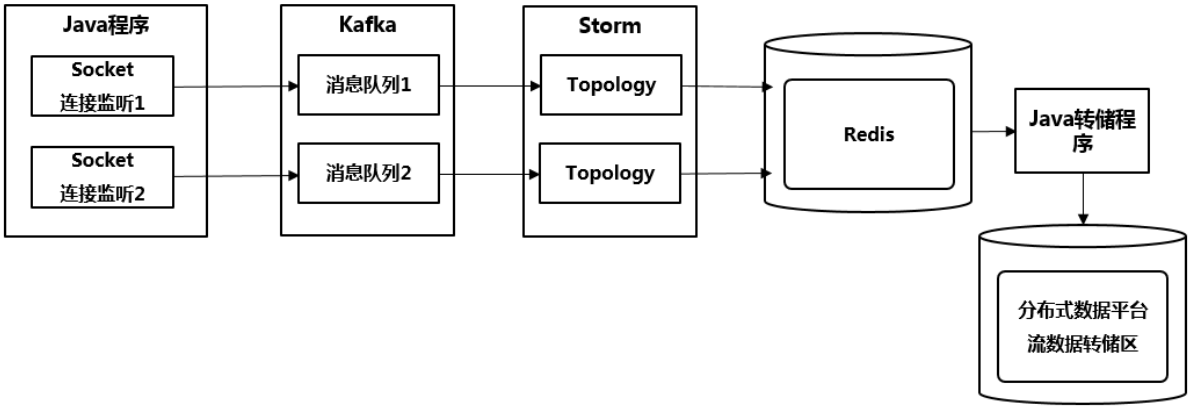


图 4-17 流数据平台整体架构图

4.5.3.2 存储内容

存储 Kafka 队列中各源系统的各种实时监测数据，流平台仅保存一定时间的数据，后续数据将周期转储到分布式数据平台中的流数据转储区中。

4.5.3.3 Kafka 设计

因为实时数据采集的速度和数据处理的速度不同步，我们采用了 Kafka 分布式消息系统，实现消息缓冲，分区处理消息，提高并发。系统只需要按照约定的报文格式把消息发送我们的 Socket 监听程序即可，而不需要关心后续的消息处理流程。

按照一条数据链路对应一个消息队列的原则，Kafka 集群中预先创建对应的消息主题 Topic，实现不同数据的并行处理，根据多数据源系统和多数据链路的实际情况，对消息主题按“数据源系统名_数据链路 ID”规则命名。

当报文接收组件接收到有效测点数据帧后，数据帧原样进入 Kafka 队列,将每个 Topic 划分了 10 个 Partition，根据消息类型值取模后，分别存储对应的的 Partition 中，提供并发能力，如图 4-18 所示：

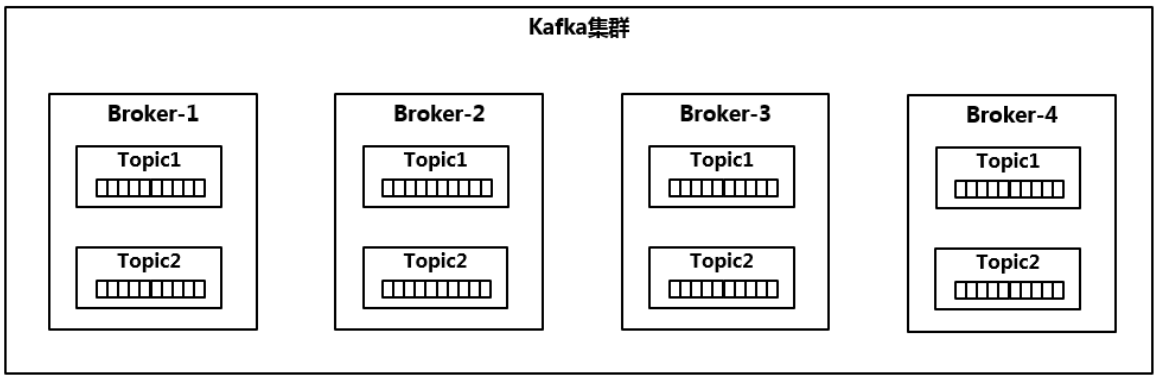


图 4-18 Kafka 集群

当消息经由我们的监听程序接收并发送到 Kafka 队列中后，源系统即可认为报文已经成功被处理。Kafka 集群会实现报文数据的持久化工作。而稍后则会由 Storm 的消息处理组件从 Topic 中按照顺序消费产生的实时消息。

4.5.3.4 Storm 设计

在 Storm 中，我们以 Topology 来作为一个基础的流计算处理单元，根据一个数据链路建立一个对应 Topic 和 Topology 的原则，每个 Topology 只处理一个 Topic 的实时消息数据，一个拓扑中包括包括 1 个 spout 和多个 bolt 角色。其中 spout 发送消息，负责将数据流以 tuple 的形式发送出去；而 bolt 则负责转换这些数据流，在 bolt 中可以完成计算、过滤等操作，bolt 自身也可以随机将数据发送给其他 bolt。

大数据基础平台中 Kafka 根据 Storm topology 在其上注册的 ID,查找对应 offset，并推送数据，spout 收到后喷发至 bolt1 进行解析的数据帧，得到里面 1 个或者多

个实时数据的值，bolt1 喷发至 bolt2，bolt2 对解析到的数据进行分片，同时将毫秒和时间片数据存储至 Redis，如图 4-19 所示：

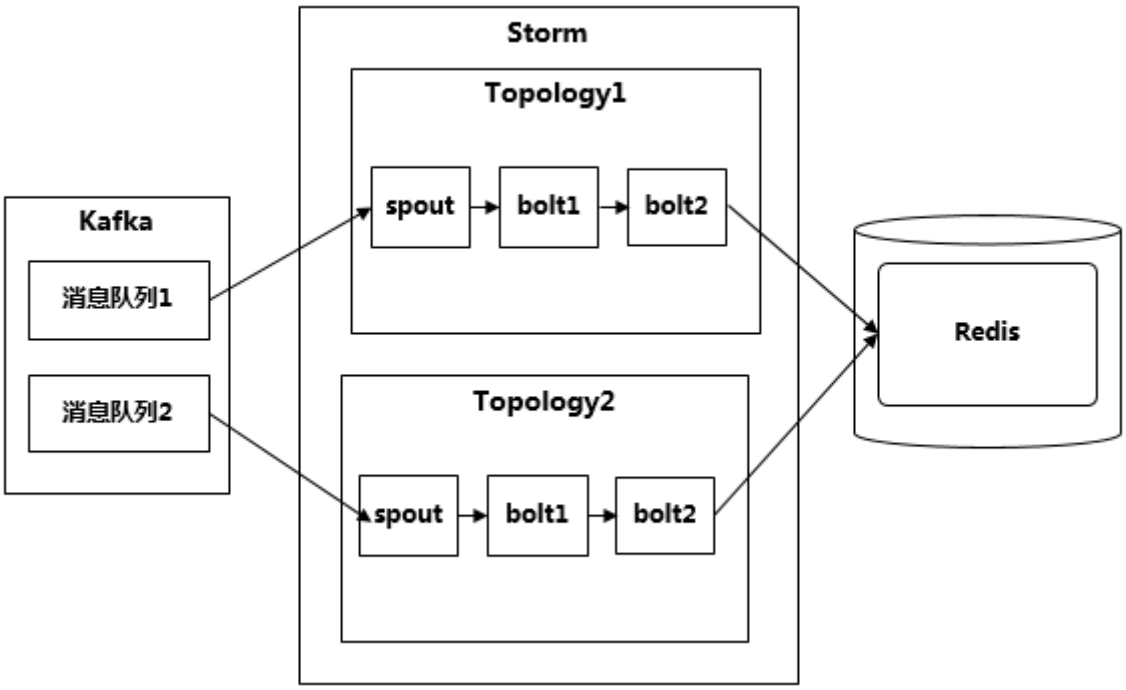


图 4-19 Storm 架构图

4.5.3.5 Redis 数据结构设计

Redis 是一个基于内存的、基于 Key-Value 结构存储的数据库，其值的数据类型支持 String 字符串、List 链表、Hash 哈希、Set 集合和 Zset 有序集合。Redis 的数据类型都支持 push/pop、add/remove、取交集并集和差集等操作，并且都是原子操作。以下分别对缓冲区数据结构的 Key 和 Value 分别进行说明：

Key 的命名规则：电站 ID|测点 ID|时间 ID，具体如表 4-15 所示：

表 4-15 Key 结构

Key 组成部分	功能描述	数据类型
电站 ID	数据来源电站的 ID	String
测点 ID	产生实时数据的测点 ID	String
时间	产生数据的时间	Date

Value 为对应的实时测点值，根据数据集成的需求分析，计算机监控系统的实时数据，缓冲数据中一个测点会保留两种类型的数据：测点的当前最新值和测点的过去半小时内的所有时间记录值，分别对应着 Redis 中的 String 和 List 和两种数

据结构，我们需要使用的 Value 的类型分别为 String 和 List。

4.5.3.6 实时数据转储模块设计

根据设计半小时将 Redis 数据转储到到 HBASE。从 Redis 获取需要进行转储的实时数据，为了避免操作 HBase 失败，导致实时数据丢失，需要先从列表中查询出需要处理的数据，操作 HBase 成功后再从列表中移除数据。当 HBase 出现异常，长时间没有恢复时，根据设置的最大列表长度，将时间最早的数据从列表中清除。

4.6 本章小结

本章依据系统需求分析，进行了大数据基础平台的设计，对平台总体框架，各种数据类型集成场景，数据存储平台的设计进行了详细的描述，并提供了各类架构图，流程图，表结构等。

第五章 系统实现与测试

5.1 平台产品选型与部署

大数据基础平台的产品选型如表 5-1 所示：

表 5-1 产品选型

序号	产品分类	产品名称	产品用途
1	数据仓库	GBase 8t	数据仓库
2	分布式数据平台	HBase 0.98	分布式列式数据库
		HDFS 2.7	分布式存储
3	流数据平台	Kafka 0.8.1	分布式消息队列
		Storm 0.9.3	实时流计算
		Redis 3.0.3	内存数据库
4	ETL 数据抽取	Kettle 5.3	ETL 数据抽取

大数据基础的硬件平台总共由 14 台服务器组成，部署情况如下图 5-1 所示，其中：

Kettle 服务器负责执行结构化数据 ETL 任务，均衡部署于两台服务器中，减轻单台服务器压力，若 ETL 任务繁重，可以横向扩展服务器。

数据仓库服务器用于存储结构化数据，采用 GBase8t 数据库产品，使用两台服务器组建 HA 集群，确保数据存储的高可用性。

分布式数据平台和流数据平台由十台服务器组成：管理节点使用两台服务器组建 HA 集群，确保 HDFS、HBase 服务的高可用性；监控节点使用一台服务器，同时考虑服务器资源的充分利用，并将内存数据库也部署在监控节点上；存储节点使用三台服务器，负责分布式数据平台的数据存储；计算节点使用四台服务器，负责流数据平台的数据处理。

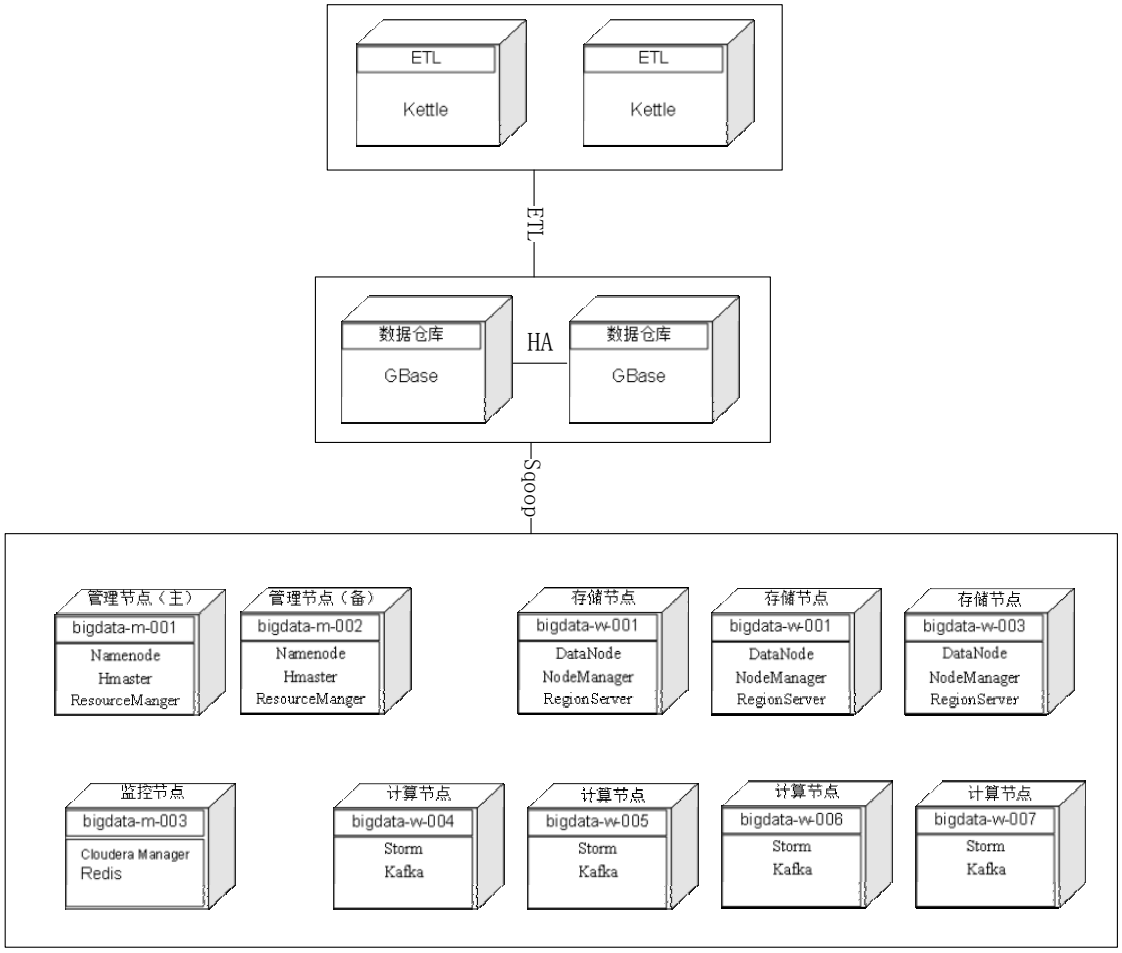


图 5-1 大数据基础平台部署图

5.2 数据仓库平台实现

5.2.1 数据仓库结构

数据仓库采用产品 GBase 8t，搭建 HA 集群，架构的集群环境采用 PRI+SDS 结构，如图 5-2 所示，其中：

PRI 为主用服务器，SDS 为共享存储备用服务器，PRI 和 SDS 均可读写，若 PRI 发生故障，SDS 将升级为 PRI。

SDS 服务器与 PRI 共享磁盘阵列，数据库 chunk 使用裸设备，避免了数据重复存储，节省了空间；

CM1、CM2 是连接管理器，在两个节点上各自配置，CM1 部署仲裁连接管理器，CM2 部署备份连接管理器。CM 负责管理来自客户端的连接请求，当 PRI 出现故障时，CM 可以将客户端的连接请求重定向到到备用服务器，实现高冗余性。

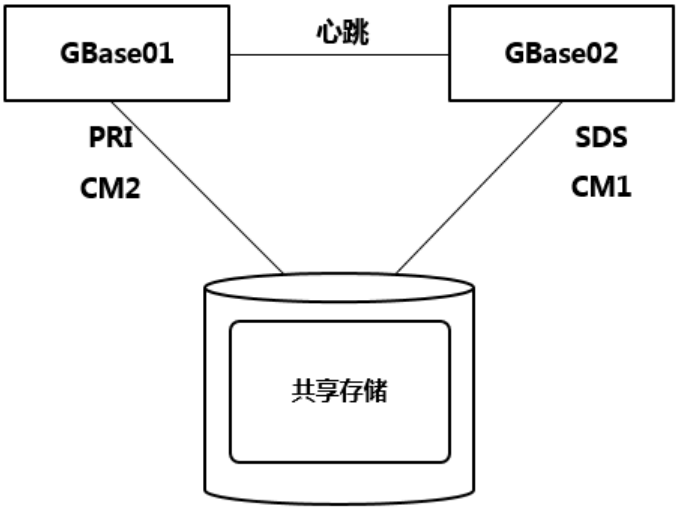


图 5-2 数据仓库 Gbase 集群架构

5.2.2 数据仓库配置

GBASE 结构设计确认后，开始 GBASE 数据库配置工作，安装完成后配置规划配置 DBSpace 和 chunk，然后开始建表工作。

5.2.3 配置 dbspace 和 chunk

根据规划建立 DBSpace 和 chunk，核心命令如下：

onspaces -c -d logdbs -p /opt/gbase/sharedbs/logchk1 -o 0 -s 153600000

onspaces -a logdbs -p /opt/gbase/sharedbs/logchk2 -o 0 -s 153600000

onspaces -c -d phydbs -p /opt/gbase/sharedbs/phychk1 -o 0 -s 51200000

onspaces -c -d tmpdbs1 -p /opt/gbase/sharedbs/tmpchk1 -o 0 -s 51200000 -k 8 -t

onspaces -c -d tmpdbs2 -p /opt/gbase/sharedbs/tmpchk2 -o 0 -s 51200000 -k 8 -t

onspaces -c -d tmpdbs3 -p /opt/gbase/sharedbs/tmpchk3 -o 0 -s 51200000 -k 8 -t

onspaces -c -d tmpdbs4 -p /opt/gbase/sharedbs/tmpchk4 -o 0 -s

51200000 -k 8 -t

onspaces -c -b sds_alt_comm -g 2k -p /opt/gbase/sharedbs/sds_alt_comm -o 0 -s 1024000

onspaces -c -S sbdbs -p /opt/gbase/sharedbs/sbchk1 -o 0 -s 102400000

onspaces -a sbdbs -p /opt/gbase/sharedbs/sbchk2 -o 0 -s 102400000

onspaces -a sbdbs -p /opt/gbase/sharedbs/sbchk3 -o 0 -s 102400000

onspaces -a sbdbs -p /opt/gbase/sharedbs/sbchk4 -o 0 -s 102400000

onspaces -a sbdbs -p /opt/gbase/sharedbs/sbchk5 -o 0 -s 102400000

onspaces -c -d odmprdspace01 -k 8 -p /opt/gbase/sharedbs/odmprdchk01 -o 0 -s 204800000

onspaces -a odmprdspace01 -p /opt/gbase/sharedbs/odmprdchk02 -o 0 -s 204800000

onspaces -a odmprdspace01 -p /opt/gbase/sharedbs/odmprdchk03 -o 0 -s 204800000

onspaces -a odmprdspace01 -p /opt/gbase/sharedbs/odmprdchk04 -o 0 -s 204800000

onspaces -a odmprdspace01 -p /opt/gbase/sharedbs/odmprdchk05 -o 0 -s 204800000

使用 onstat -d 命令查看 DBSpace 与 chunk 的配置使用情况，如图 5-3 和图 5-4 所示：

```
$onstat -d

GBase 8t  Server Version 12.10.FC4G1AEE -- Read-Only (SDS) -- Up 00:02:19 -- 65165268 Kbytes

Dbspaces
address          number   flags      fchunk   nchunks  pgsize   flags    owner    name
a72e4028         1        0x60801    1        1        2048     NL BA    informix rootdbs
a9cb4830         2        0x60801    2        2        2048     NL BA    informix logdbs
a9cb4a60         3        0x60801    3        1        2048     NL BA    informix phydbs
a9cb4c90         4        0x260001   4        1        8192     N WBA    informix tmpdbs1
a9edb028         5        0x260001   5        1        8192     N WBA    informix tmpdbs2
a9edb258         6        0x260001   6        1        8192     N WBA    informix tmpdbs3
a9edb488         7        0x260001   7        1        8192     N WBA    informix tmpdbs4
a9edb6b8         8        0x60811    8        1        4096     NLBBA    informix sds_alt_comm
a9edb8e8         9        0x60801    9        10       8192     NL BA    informix odmprdspace01
a9edbb18         10       0x60801    19       10       8192     NL BA    informix fdmprdspace01
a9edbd48         11       0x60801    29       10       8192     NL BA    informix admprdspace01
a9edd028         12       0x60801    40       5        2048     NLSBA    informix sbdbs
aa1c0058         2047     0x142001   32766    1        8192     N TBA    informix sdstmp_gbase2_1
 13 active, 2047 maximum

Note: For BLOB chunks, the number of free pages shown is out of date.
      Run 'onstat -d update' for current stats.
```

图 5-3 DBSpace 配置情况

```
Chunks
address          chunk/dbs     offset     size       free       bpages     flags pathname
a72e4258         1      1      0          5120000    5057972               PI-B-- /opt/gbase/sharedbs/rootdbs
a9edd258         2      2      0          76800000   76799947              PI-B-- /opt/gbase/sharedbs/logchk1
a9ede028         3      3      0          25600000   25599947              PI-B-- /opt/gbase/sharedbs/phychk1
a9edf028         4      4      0          6400000    6399947               PO-B-- /opt/gbase/sharedbs/tmpchk1
a9ee0028         5      5      0          6400000    6399947               PO-B-- /opt/gbase/sharedbs/tmpchk2
a9ee1028         6      6      0          6400000    6399947               PO-B-- /opt/gbase/sharedbs/tmpchk3
```

图 5-4 chunk 配置情况

5.3 分布式数据平台和流数据平台实现

5.3.1 硬件规划

本次大数据基础平台的分布式数据平台和流数据平台共由 10 台服务器组建，规划部署角色如表 5-2 所示：

表 5-2 服务器部署角色规划

组件分类	机器名	部署角色	安装组件
分布式平台	bigdata-m-001	管理节点（主）	Namenode HDFS 管理 Hmaster HBASE 管理 ResourceManger 资源管理
	bigdata-m-002	管理节点（备）	Namenode Hmaster ResourceManger
	bigdata-m-003	监控节点	Cloudera Manager Redis
	bigdata-w-001	存储节点	DataNode NodeManager RegionServer HBASE 的数据
	bigdata-w-002	存储节点	DataNode NodeManager RegionServer
	bigdata-w-003	存储节点	DataNode NodeManager RegionServer
	bigdata-w-004	流计算节点	Storm Kafka
	bigdata-w-005	流计算节点	Storm Kafka
	bigdata-w-006	流计算节点	Storm Kafka
	bigdata-w-007	流计算节点	Storm Kafka

5.3.2 平台安装配置

1.按规划将 IP 与服务器名进行关联，并配置到集群中每台服务器的/etc/hosts，并查看确认，如图 5-5 所示：

```
# vi   /etc/hosts
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1          localhost localhost.localdomain localhost6 localhost6.localdomain6
10.185.35.6      bigdata-m-001.bigdata.com
10.185.35.7      bigdata-m-002.bigdata.com
10.185.35.1      bigdata-m-003.bigdata.com
10.185.35.21     bigdata-w-001.bigdata.com
10.185.35.23     bigdata-w-002.bigdata.com
10.185.35.25     bigdata-w-003.bigdata.com
10.185.35.10     bigdata-w-004.bigdata.com
10.185.35.11     bigdata-w-005.bigdata.com
10.185.35.12     bigdata-w-006.bigdata.com
10.185.35.13     bigdata-w-007.bigdata.com
```

图 5-5 hosts 配置

2.配置大数据软件源

在 bigdata-m-003 上传相关 CDH、cm5 软件，编辑 bigdata.repo 文件，并传送到每台机器上的/etc/yum.repos.d/目录下，bigdata.repo 的配置内容如下：

#vi /etc/yum.repos.d/bigdata.repo

[cloudera-manager]

name = 大数据基础运行平台, Version 5.7.0

baseurl = http://bigdata-m-003/cm5/5.7.0

gpgcheck = 0

3.安装大数据基础运行平台服务

安装 cm 使用的元数据库，并确认数据库配置成功后启动 cloudera 服务，主要配置如下：

/usr/share/cmf/schema/scm_prepare_database.sh -h bigdata-m-001-P 3306 mysql scm Hadoop password

4.安装大数据基础运行组件

访问大数据基础运行平台地址，http://bigdata-m-003:7180，为规划的服务器安

装 CDH，建立群集，如图 5-6 所示：

图 5-6 为 CDH 群集安装指定主机

使用 parcel 代替软件包进行安装，CDH,Kafka,Redis,Storm 等组件加入 parcel 库，并安装，如图 5-7 所示：

图 5-7 利用 parcel 安装软件

配置服务器角色，数据库设置，输入在 mysql 中创建的数据库，测试连接，如图 5-8 所示：

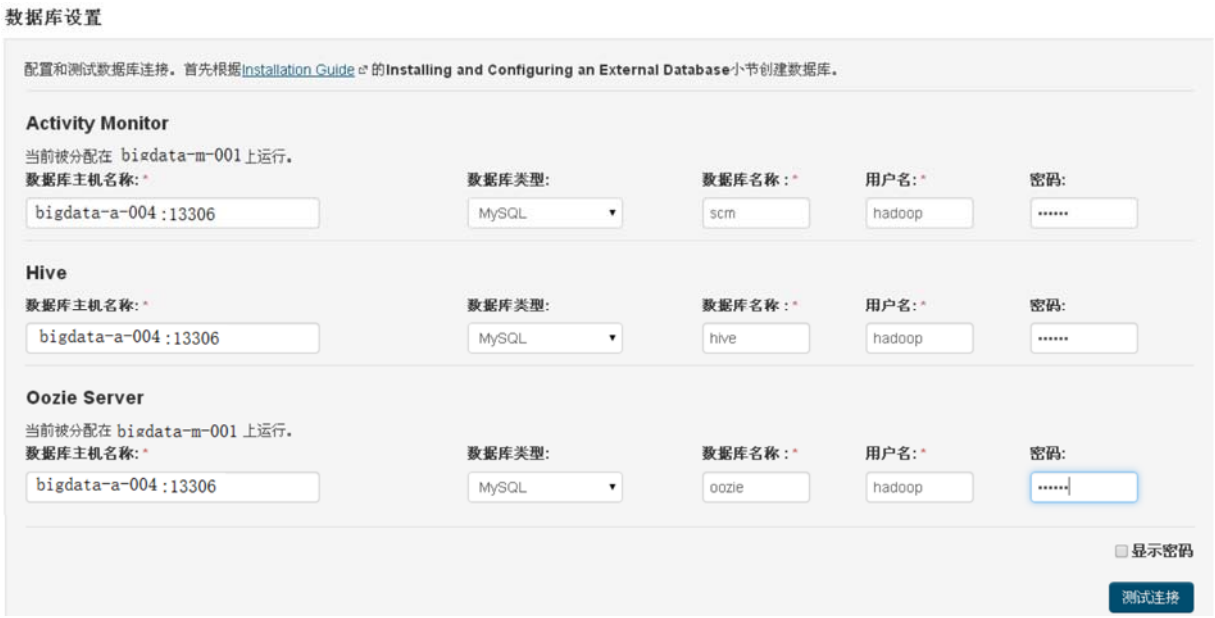


图 5-8 数据库设置

安装成功后，自动启动服务，可访问 cloudera manager。cloudera manager 可对集群进行管理和监控，并对 HADOOP 的多组件进行了整合，如图 5-9 所示：

图 5-9 cloudera manager 管理群集

5.HA 配置

根据硬件规划，NameNode 主节点为 bigdata-m-001。bigdata-m-002 作为 Standby 的节点,配置如图 5-10 所示：

图 5-10 启用 HA

6.安装组件

根据硬件规划，选择对应服务器，添加相应的 HBASE,Kafka，Storm，Redis 等组件，本文不再详诉。

5.4 结构化数据集成实现

结构化数据的的集成采用开源的 ETL 工具 Kettle 实现，根据源信息系统的要求，分为了大数据平台提供接口表或者由源系统提供结构表的两种场景。下面以公司电力生产管理信息系统为例，介绍结构化数据集成的具体实现，具体实现如下：

源系统与数据仓库连接建立后，读取 Kettle 资料库中 KJB_EPPMS_TO_ODM_CFG，获取需抽取表名，表之间转换的对应关系等配置信息，并读取关系型数据推送日志 etl_table_log，判断周期内是否有数据推送成功，配置如图 5-11 所示：

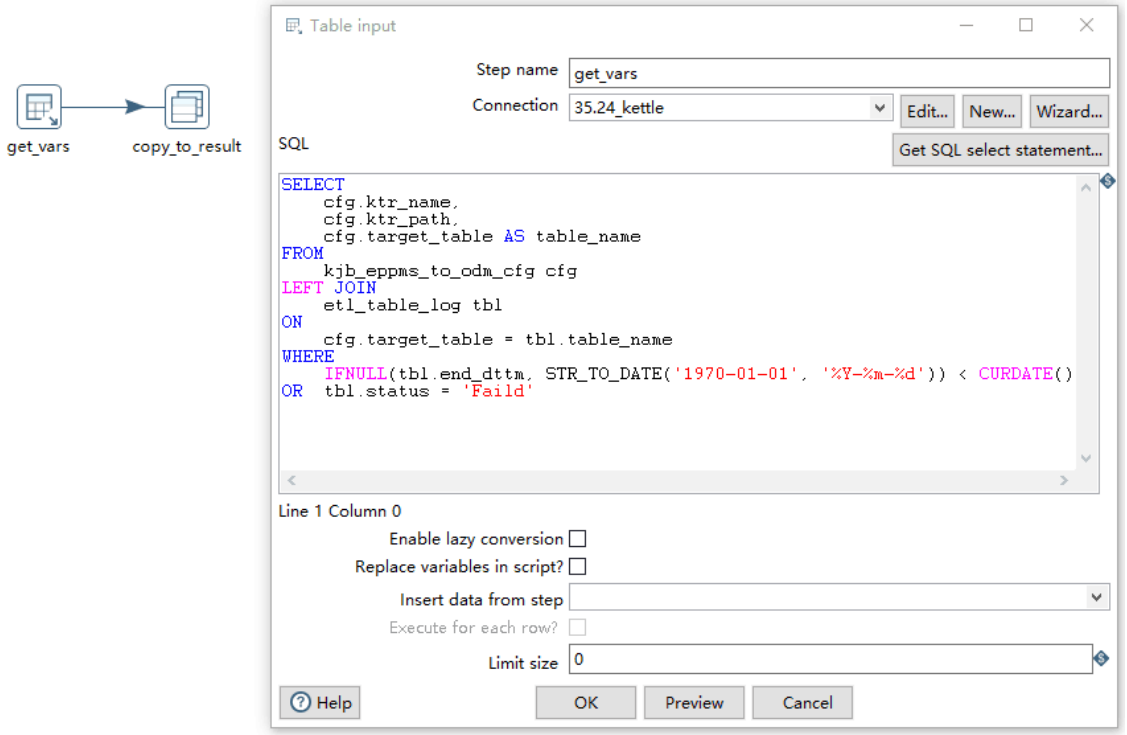


图 5-11 读取作业配置信息并判断作业是否执行

判断完成后 ETL 开始集成源系统结构化数据数据至数据缓冲区，流程如图 5-12 所示：

图 5-12 结构化数据集成流程图

抽取开始首先删除 3 天之前的数据，保留 3 天历史数据，用于数据出错时的排查，配置如图 5-13 所示：

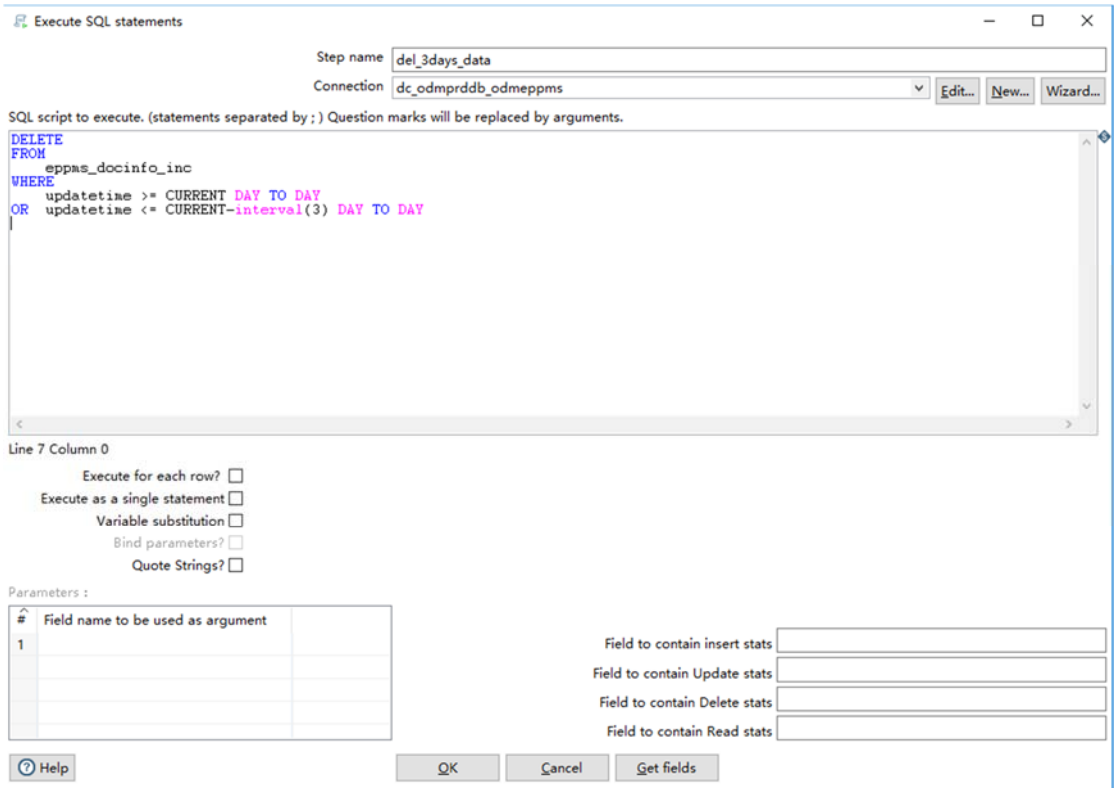


图 5-13 del_3days_data 配置

全量获取源系统数据表，配置如图 5-14 所示：

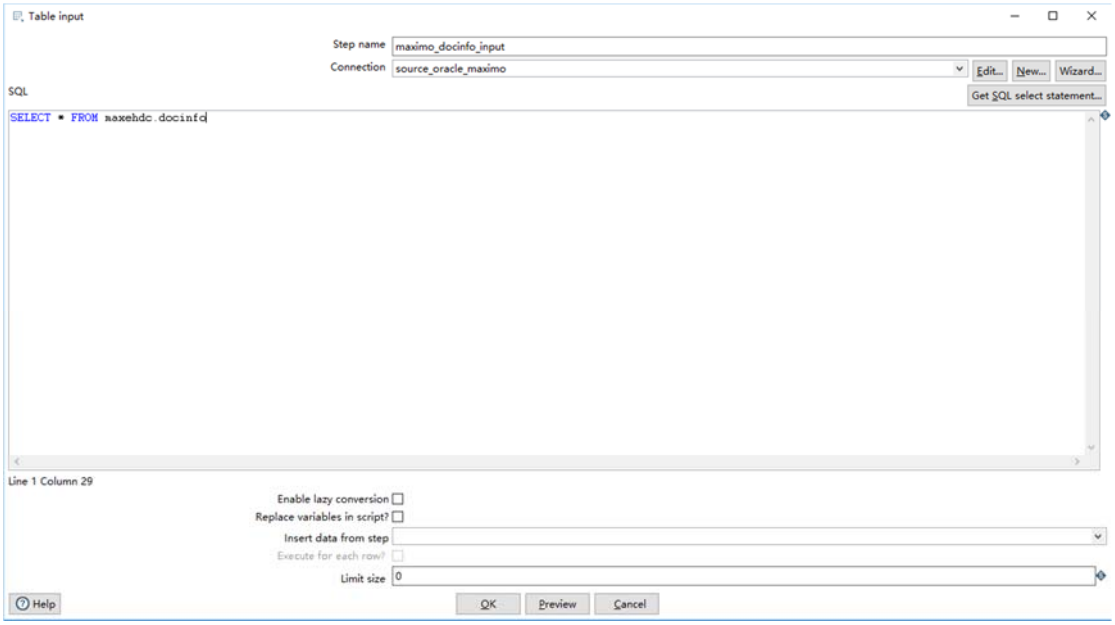


图 5-14 maximo_docinfo_input 配置

添加缓冲区的审计字段,配置如图 5-15 所示：

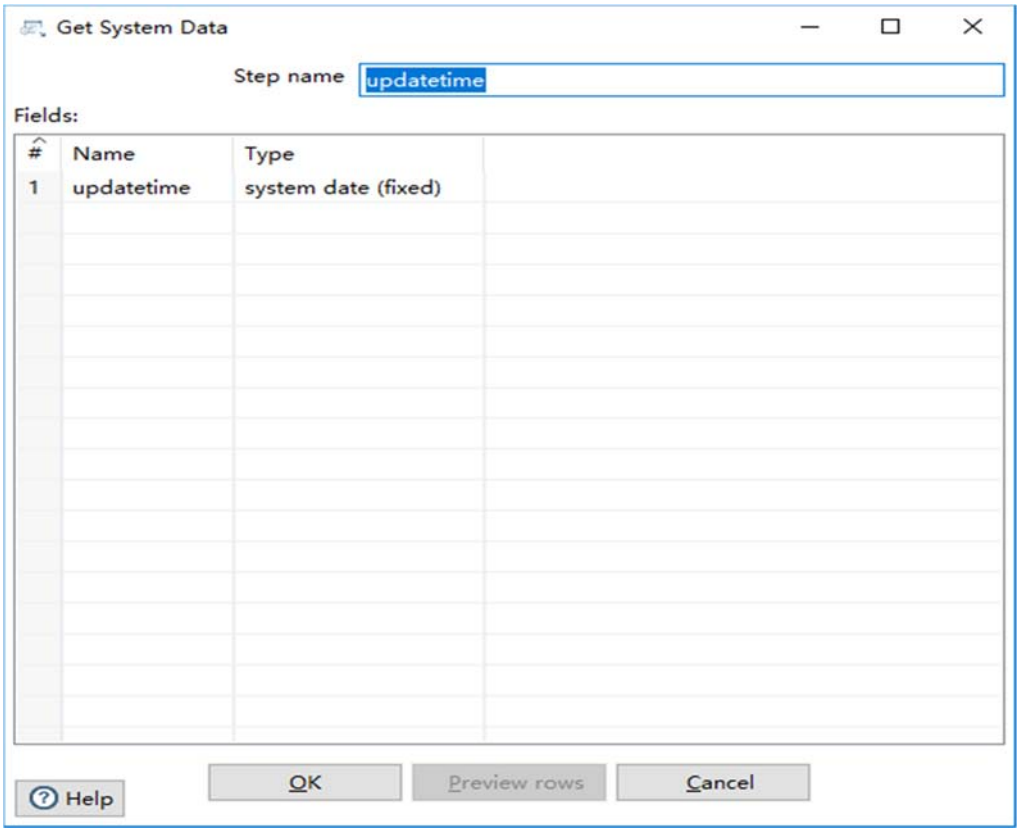


图 5-15 updatetime 配置

提取源系统数据表字段和审计字段，配置如图 5-16 所示：

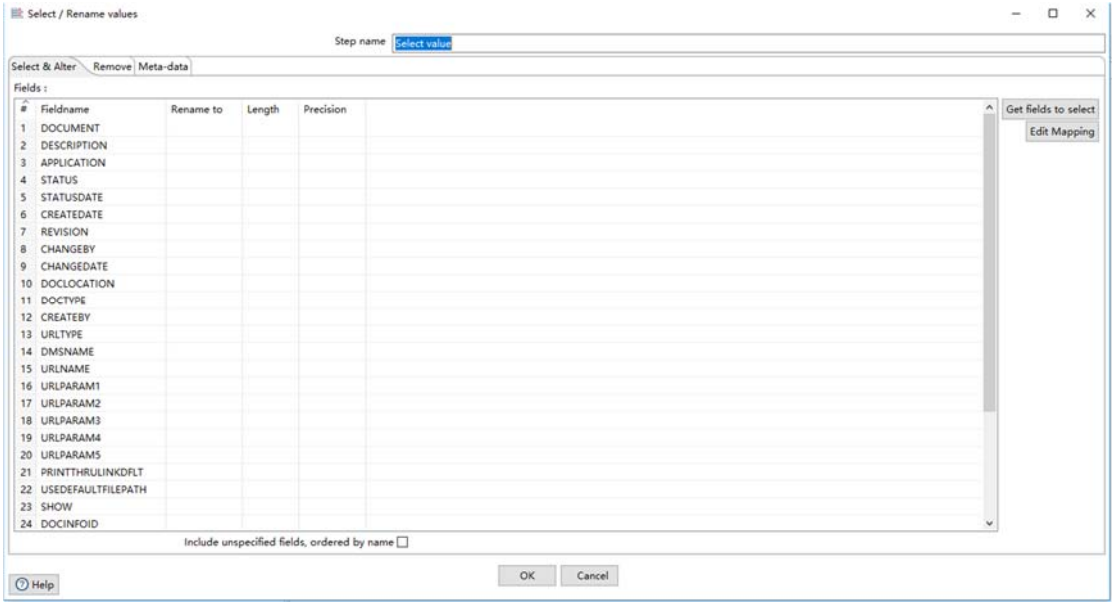


图 5-16 select value 配置

将数据写入数据仓库相应的缓冲区数据表中，配置如图 5-17 所示：

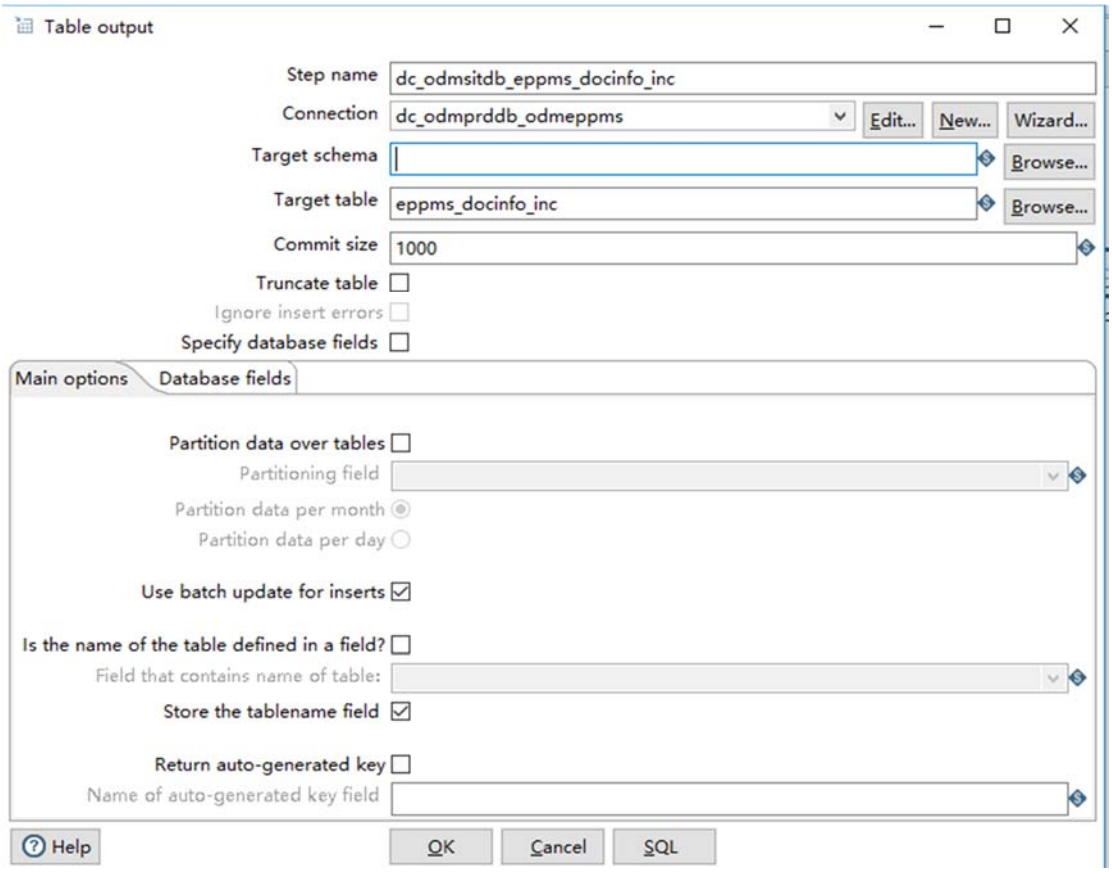


图 5-17 dc_odmsitdb_eppms_docinfo_inc 配置

5.5 ETL 作业控制调度实现

ETL 作业调度采用 Linux 操作系统自带的调度程序 Crontab，基于设计原则，为避免相同作业同时运行，需在作业运行前进行判断，shell 脚本如下：

```
ProcessNum=`ps -ef | grep job_xxx_to_yyy | grep -v grep | wc -l`
if [ ${ProcessNum} -le 2 ]; then
    # 启动作业…
fi
```

ETL 作业内部逻辑控制实现具体流程如图 5-18 所示：

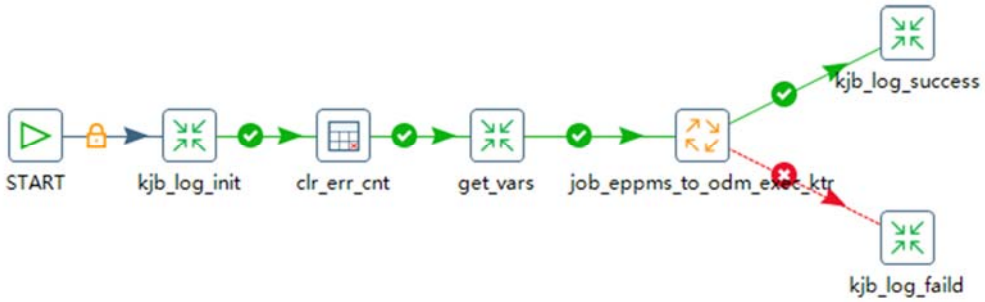


图 5-18 ETL 作业内部逻辑控制流程

具体步骤如下：

1．ETL 程序的开始入口；

2．kjb_log_init 记录作业的开始时间，运行状态为“Running”；

3．clr_err_cnt 清空错误记录表；

4．get_vars 读取作业配置表和依赖关系表中的信息，得到满足更新条件的数据表及其对应的转换名称和转换路径；

5．执行数据转换的 job；

6．job 成功后 kjb_log_success 记录作业的结束时间，运行状态为“Finish”，；

7．job 执行失败后 kjb_log_faild 记录作业的结束时间，运行状态为“Failed”。

执行数据转换的 JOB 的具体流程如图 5-19 所示：

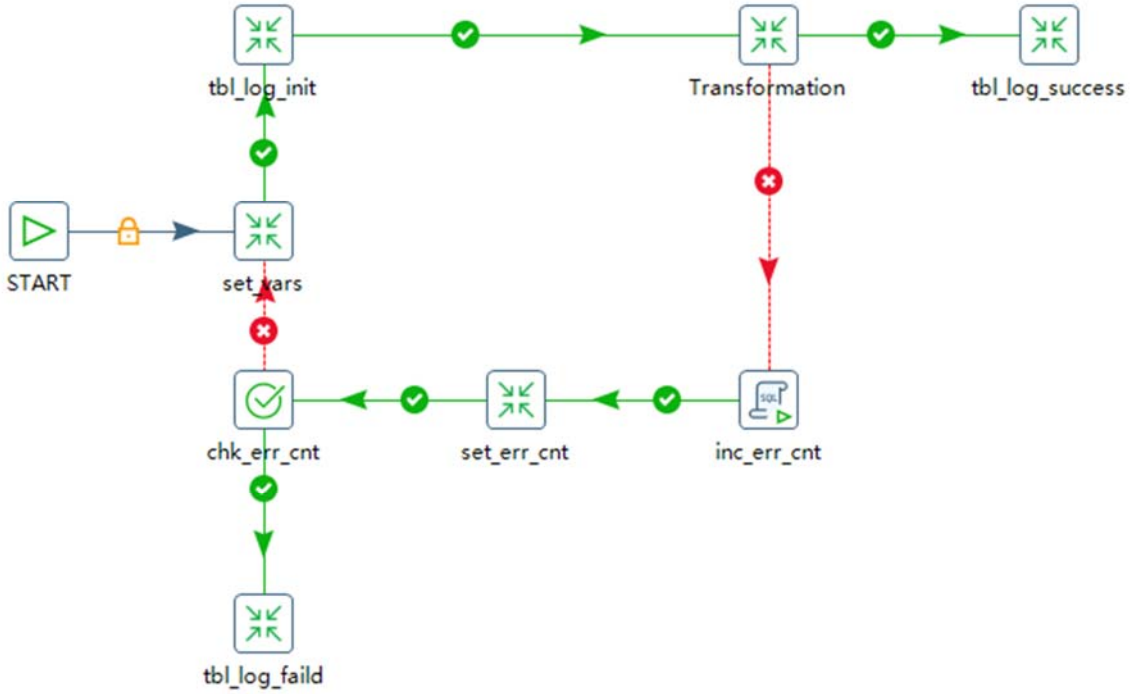


图 5-19 ETL 数据转换控制调度流程

具体步骤如下：

1．数据更新作业的开始入口；

2．根据读取到的信息设置变量 set_vars；

3．tbl_log_init 记录转换的开始时间，运行状态为“Running”；

4．执行转换；

5. 当 4 执行成功后 tbl_log_sucess 记录转换的结束时间，运行状态为“Finish”；

6．当 4 执行失败后记录当前失败的数据表名称；

7．统计当前数据表失败的次数 set_err_cnt；

8．chk_err_cnt 若失败次数大于等于 10 次则进入 9，否则返回 2；

9．tbl_log_faild 记录转换的结束时间，运行状态为“Failed”。

5.6 非结构化数据采集

非结构化数据采集主要采用了接口数据文件和接口调用两种方式，下面分别两种方式的具体实现。

5.6.1 接口数据文件方式

接口数据文件方式方式时序图如图 5-20 所示：

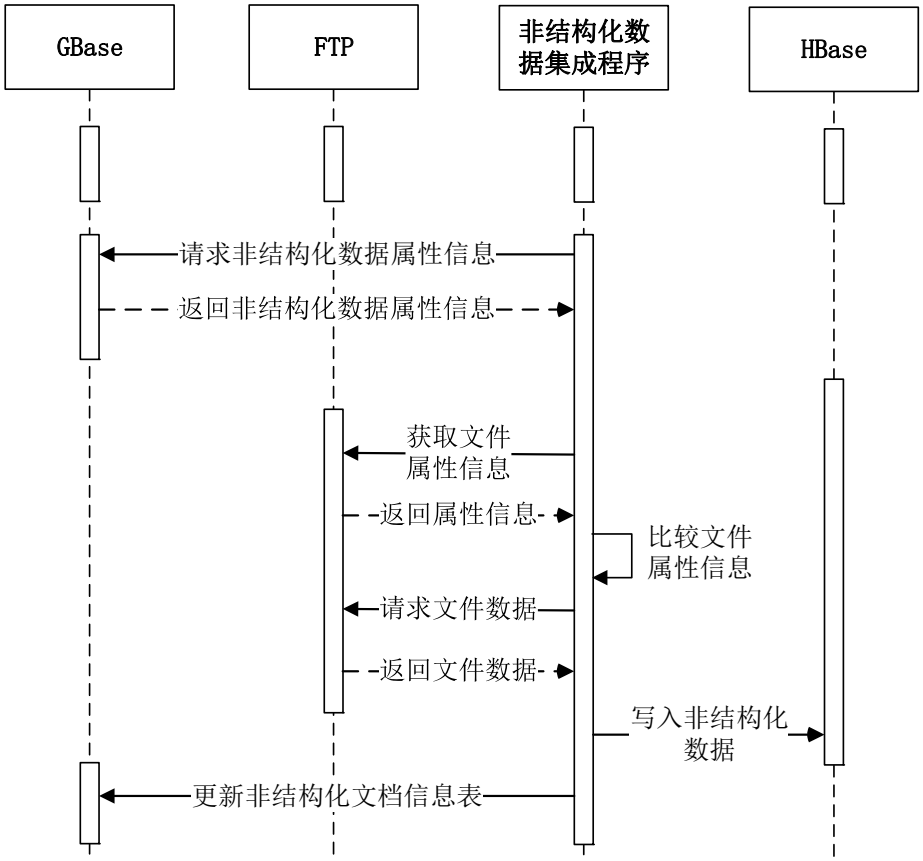


图 5-20 数据接口文件方式时序图

接口数据文件方式获取非结构化数据的核心代码如下：

```
//获取非结构化文档日志表信息
private void getGbaseList() {
    try {
        Class.forName("com.informix.jdbc.IfxDriver");
        connnection = DriverManager.getConnection(dbUrl, dbUser, dbPass);
        String exists = "SELECT sysid, CASE WHEN document_id IS NOT NULL THEN 'PDMS_' || sysid || '_' || document_id ELSE 'PDMS_' || sysid || '_FID_' || file_id END AS unstruct_id,
```

```
file_id, document_id, document_name, document_path, creation_date FROM wpms_tzhp_xzp_content_file_inc;";
            ps = connnection.prepareStatement(exists);
            rs = ps.executeQuery();
            existMap = new HashMap<String,String>();
            while (rs.next()) {
                if (StringUtils.isEmpty(rs.getString("document_id"))) {
                    String document_path = rs.getString("document_path");
                    String file_size = rs.getString("file_size");
                    String create_time = rs.getString("create_time");
                    String filename = document_path.split("/")[document_path.split("/").length - 1];
                    existMap.put(StringUtils.lowerCase(filename),
rs.getString("unstruct_id")+"|"+rs.getString("file_id")+"|"+file_size +"|"+create_time);

                } else {
                    String type = "";
                    if (!StringUtils.isEmpty(rs.getString("document_path"))) {
                        String[] str = rs.getString("document_path")
                                .split("/")[rs.getString("document_path").split("/").length -
1].split("\\.");
                        if (str.length > 1) {
                            type = "." + StringUtils.lowerCase(str[str.length - 1]);
                        }
                        String document_id = rs.getString("document_id");

                        String filename = document_id + type;
                        String file_size = rs.getString("file_size");
                        String create_time = rs.getString("create_time");
                        String filename = document_path.split("/")[document_path.split("/").length
- 1];
                        existMap.put(StringUtils.lowerCase(filename),
rs.getString("unstruct_id")+"|"+rs.getString("file_id")+"|"+file_size +"|"+create_time);
```

```
                    } else {
                        String file_size = rs.getString("file_size");
                        String create_time = rs.getString("create_time");
                        String filename = document_path.split("/")[document_path.split("/").length
- 1];
                        existMap.put(StringUtils.lowerCase(filename),
rs.getString("unstruct_id")+"|"+rs.getString("file_id")+"|"+file_size +"|"+create_time);

                    }
                }
            }
            System.out.println("总文件数 : " + existMap.size() + "  个");
        }
}
```

//比较并获取 FTP 文件

```
private void getFileList(String path) {
    try {
        FTPFile[] ftpFiles = ftpClient.listFiles(path);
        for (FTPFile ftpFile : ftpFiles) {
            if (ftpFile.isDirectory()) {
                String nextPath = path + ftpFile.getName() + "/";
                getFileList(nextPath);
            } else {
                String fileName = StringUtils.lowerCase(ftpFile.getName());
                if (existMap.containsKey(fileName)) {
                    String[] filenamewithdot = ftpFile.getName().split("\\.");
                    FileInfo fileInfo = new FileInfo();
                    if (filenamewithdot.length > 1) {
                        fileInfo.setUnstruct_id(filenamewithdot[0]);
                        fileInfo.setFile_type(filenamewithdot[1]);
                    } else {
                        fileInfo.setFile_type("");
                    }
```

```
                    String str = existMap.get(fileName);
                    String[] strings = str.split("\\|");
                    if(strings.length > 2){
                        fileInfo.setUnstruct_id(strings[0]);
                        fileInfo.setFile_id(Long.valueOf(strings[1]));
                        fileInfo.setDocument_id(Long.valueOf(strings[2]));
                    }else{
                        fileInfo.setUnstruct_id(strings[0]);
                        fileInfo.setFile_id(Long.valueOf(strings[1]));
                    }
                    fileInfo.setDocument_name(ftpFile.getName());
                    fileInfo.setDocument_path(path + ftpFile.getName());
                    fileInfo.setFile_size(ftpFile.getSize());

    fileInfo.setCreation_date(String.valueOf(ftpFile.getTimestamp().getTimeInMillis()));

    fileInfo.setMd5_unstruct_id(MD5Util.Md5(existMap.get(ftpFile.getName())));
                    System.out.println(fileInfo);

                    for (String key : existMap.keySet()) {
                        if(key.equals(fileInfo.getMd5_unstruct_id())){
                            String[] str = existMap.get(key).split("|");
                            if(!str[3].equals(fileInfo.getFile_size())
|| !str[4].equals(fileInfo.getCreation_date())){
                                String deletesql = "delete from " + INSERT_G_TABLE + "
where unstruct_id='" + rs.getString("unstruct_id");
                                psdeleteold = connnection.prepareStatement(deletesql);
                                psdeleteold.executeUpdate();
                            }
                        }
                    }
```

```
                    insertFileToHBase(fileInfo);
                }
            }
        }
    }
}
    //将非结构化数据写入 HBASE
private void insertFileToHBase(FileInfo fileInfo) {
    System.setProperty("Hadoop.home.dir", "D:\\java\\Hadoop");
    HFileOutputStream out = null;
    try {
        HFile hfile = new HFile(fileInfo.getMd5_unstruct_id(), fileInfo.getDocument_name(),
fileInfo.getDocument_path(),
                fileInfo.getUnstruct_id(), String.valueOf(fileInfo.getFile_size()),
fileInfo.getFile_type(), fileInfo.getCreation_date());
        out = new HFileOutputStream(hfile, H_TABLE);
        ftpClient.retrieveFile(fileInfo.getDocument_path(), out);
        insertFileInfoGbase(fileInfo);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (null != out)
                out.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5.6.2 接口调用方式

通过文档信息表中的信息判断文件是否有变化，如果有就通过源系统提供的接口依照文件 ID 去读取文件，实现将源业务系统增量的非结构化数据存放到分布

式数据平台中，接口数据文件方式时序图如图 5-21 所示：

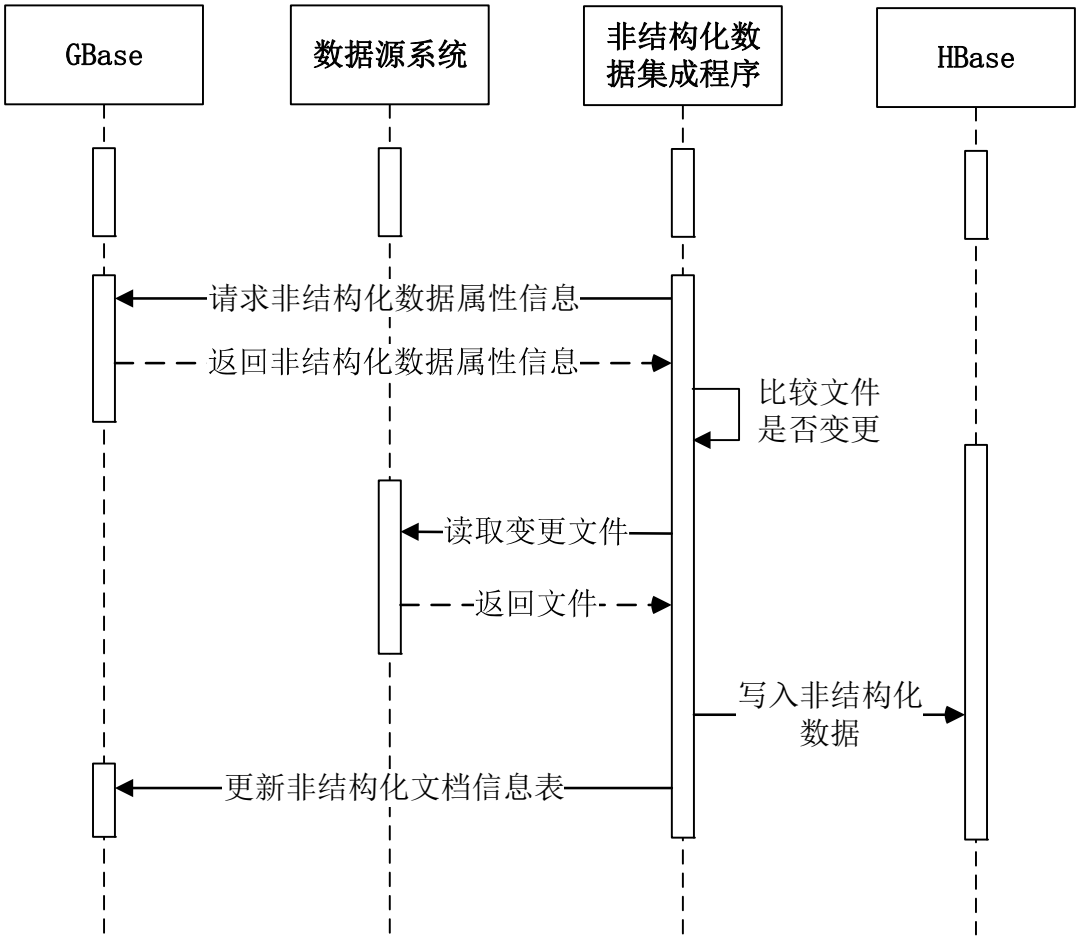


图 5-21 接口调用方式时序图

核心代码如下：

```
while( rs.next() ){
    String documentId = null;
    try {
     //获取文档 ID
        documentId = rs.getString( 2 );
        ISCSDocumentID CISDocId = icisApp.getUCPMAPI ().getActiveAPI
().createActiveDocumentID( documentId );
        ISCSFileAPI getfileApi = icisApp.getUCPMAPI ().getActiveAPI ().getFileAPI ();
        ICISTransferStream fileResult = getfileApi.getFile( userContext , CISDocId );
        if( null != fileResult ){
```

```
            InputStream is = null;
            HFileOutputStream hout = null;
            try {
    //获取文档数据流
                is = fileResult.getInputStream();
                String fileId = "PDMS_" + sysId + "_" + documentId ;
                String planFileId = fileId;
                String fileName = fileResult.getFileName();
                String MD5ID =  MD5Util.Md5( fileId );
                String fileType = "";
                if( -1 != fileName.indexOf( "." ) ){
                    fileType = fileName.substring( fileName.lastIndexOf( "." ),
fileName.length() );
                }
                long fileLength = 0;
                byte[] buf = new byte[1024];
                int i = is.read( buf );
                HFile hFile = new HFile( MD5ID , fileName , "" , planFileId , "0",
                        fileType , df.format( new Date() ) );
    //写入 HBASE
                hout = new HFileOutputStream( hFile ,
HFileConfiguration.storeTableName_PDMS );
                if( -1 != i ){
                    hout.write( buf , 0 , i );
                    fileLength += i;
                }
                while( -1 != ( i = is.read( buf ) ) ){
                    hout.write( buf ,  0 , i );
                    fileLength += i;
                }
                hout.flush();
                hFile.updateFileSize( HFileConfiguration.storeTableName_PDMS ,
fileLength );
```

```
        try{ pstm1.clearParameters(); }catch( Exception e ){}
        pstm1.setString( 1 , planFileId );
        rs1 = pstm1.executeQuery();
        boolean isExist = false;
        if( rs1.next() ){
            if( rs1.getInt( 1 ) > 0 )
                isExist = true;
        }
        try{ rs1.close(); }catch( Exception e ){}
        if( isExist ){
            pstm = pstmUpd;
        }else{
            pstm = pstmIns;
        }
        try{ pstm.clearParameters(); }catch( Exception e ){}
        pstm.setString( 1 , planFileId );
        pstm.setLong( 2 , fileLength );
        pstm.setString( 3 ,    fileType );
        pstm.setTimestamp( 4 ,    new java.sql.Timestamp( new Date().getTime() ) );
        pstm.setString( 5 ,    MD5ID );
        if( isExist ){
            pstm.setString( 6 ,    planFileId );
        }

        pstm.executeUpdate();
        iii ++;
        logger.debug( "store->id:{},name:{} success" , planFileId , fileName );
    } catch (IOException e) {
        failLogger.info( "store->sysId:{},documentId:{} fail" , sysId , documentId );
        failLogger.info( "ex trace" ,    e );
    }finally{
        try{ if( null != is ){is.close();} }catch( Exception e ){}
        try{ if( null != hout ){ hout.close();}}catch( Exception e ){}
```

```
            }
        }else{
            failLogger.info( "file-> documentid:{} not found" , documentId );
        }
    } catch (CommandException e) {
        failLogger.info( "store->sysId:{},documentId:{} fail" , sysId , documentId );
        failLogger.info( "ex trace" ,   e );
    }
}
```

5.7 实时数据集成实现

实时数据集成时序图如图 5-22 所示：

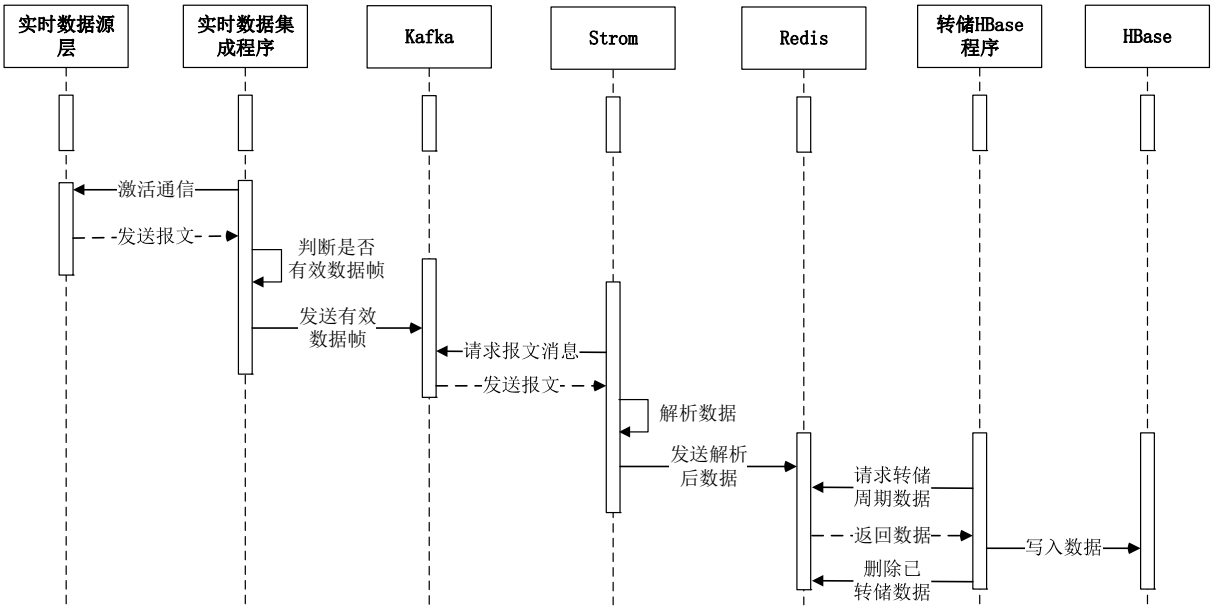


图 5-22 实时数据集成时序图

1.创建 socket 连接，连接到源系统数据交换平台，核心代码如下：

```
public void run() {
    if(socket.isClosed()){
        try {
            if(flag[0] == 0){
                socket = new Socket("10.186.50.14", 15001);
                socket.setKeepAlive(true);
```

```
                    Client104.doconnect(socket, 1);
                    System.out.println(DateUtil104.getSystemDate() + " -10.186.50.14-
socket<15001> closed, open!");
                }else{
                    socket = new Socket("10.186.50.13", 15001);
                    socket.setKeepAlive(true);
                    Client104.doconnect(socket, 1);
                    System.out.println(DateUtil104.getSystemDate() + " -10.186.50.13-
socket<15001> closed, open!");
                    constartTime = System.currentTimeMillis();
                }
            }
        }
}
```

2. 配置 Kafka 连接信息，判断接收到的实时数据帧是否为有效数据帧，判断完成后将有效数据帧发送 Kafka 消息队列中，核心代码如下：

```
public static void doconnect(Socket socket, int address) {
          Properties props = new Properties();
           props.put("metadata.broker.list",
"bigdata-w-004.bigdata.com:9092,bigdata-w-005.bigdata.com.com:9092" +
                    ",bigdata-w-006.bigdata.com:9092,bigdata-w-007.bigdata.com:9092");
           props.put("serializer.class", "kafka.serializer.StringEncoder");
           props.put("key.serializer.class", "kafka.serializer.StringEncoder");
           props.put("partitioner.class", "com.ylj.storm.Partitioner104");
           props.put("request.required.acks", "1");
           ProducerConfig config = new ProducerConfig(props);
           producer = new Producer<String, String>(config);
……
for (int j = 0; j < list.size(); j++){
    pack_per = list.get(j);
    packstr = FileType104.packToHexString(pack_per);
    String[] ss = packstr.split(" ");
    // 判断返回帧是否正常，无效帧则跳过
```

```
if(ss.length < 4){
    System.out.println("pack is error !");
    continue;
}
s2 = ss[2];
s3 = ss[3];
String type = FileType104.packConfirm(pack_per);
if("lianluConfirm".equals(type)){ //判断是否是连接激活确认帧
    System.out.println(DateUtil104.getSystemDate() + "lianluConfirm Recv LinkNo 1 <== " + packstr + " |no <== " + ss[2] + " " + ss[3]);
}else if("zongzhaohuanConfirm".equals(type)){ //判断是否是总召帧
    System.out.println(DateUtil104.getSystemDate() + " zongzhaoC Recv LinkNo 1 <==  " + packstr);
}else{
    // 针对数据帧类型返回相应的响应帧
    if (pack_per.length > 6) {
        if(ss.length > 8 && "14".equals(ss[8])){
            if(zz == 8){
                zz = 0;
                byte[] pack_or = new byte[] { 0x68, 0x04, 0x01, 0x00, (byte) Integer.parseInt(ss[2], 16), (byte) Integer.parseInt(ss[3], 16) };
                out.write(pack_or);
                out.flush();
            }
            zz++;
        }else{
            byte[] pack_or = new byte[] { 0x68, 0x04, 0x01, 0x00, (byte) Integer.parseInt(ss[2], 16), (byte) Integer.parseInt(ss[3], 16) };
            out.write(pack_or);
            out.flush();
        }
        //判断是否是心跳帧
        String point = isHeartbeatPoint(pack_per);
```

```
                if(StringUtils.isNotBlank(point)){
                }else{
                // 将数据帧写入 Kafka
                    StormProducer.getInstance().putProducer(packstr, pack_per, producer, address);
                }
            }
        }
}
```

StormProducer 的核心代码如下:

```
public void putProducer( String producer_, int[] pack_per, Producer<String, String> producer, int address){
        int partitionerNum = new Random().nextInt(10);
        if(partitionerNum >= 0){
            if(address == 3){
                count++;
                KeyedMessage<String, String> newdata = new KeyedMessage<String, String>("pack_3_15003", String.valueOf(partitionerNum), producer_);
                producer.send(newdata);
            }else if(address == 1){
                count++;
                KeyedMessage<String, String> newdata = new KeyedMessage<String, String>("pack_1_15001", String.valueOf(partitionerNum), producer_);
                producer.send(newdata);
            }
            if(count % 1000 == 0)
                System.out.println(new Timestamp(System.currentTimeMillis()) + ": message count = " + count);
        }
    }
```

3.启动一个 Strom Topology ， 一对一处理 Kafka 的 topic 中的消息

```
public class Topology_1_15001 {
    public static void main(String[] args) throws AlreadyAliveException, InvalidTopologyException, AuthorizationException {
```

```
        int storm_work_num = 4;
        int kafka_max_size = 1000;
        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("spout_1_15001", new KafkaPullSpout("pack_1_15001",
"storm_1_15001", kafka_max_size, false), 10);
        builder.setBolt("myBolt_1_15001", new
MyBolt_15001(),10).shuffleGrouping("spout_1_15001");
        builder.setBolt("secondBolt_1_15001", new
MySecondBolt_15001(),10).shuffleGrouping("myBolt_1_15001");
        Config conf = new Config();
        conf.setMessageTimeoutSecs(600);
        conf.put(Config.TOPOLOGY_MESSAGE_TIMEOUT_SECS, 600);
        conf.put(Config.SUPERVISOR_WORKER_TIMEOUT_SECS, 60);
        if (args != null && args.length > 0) {
            conf.setNumWorkers(storm_work_num);
            StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
        }
    }
```

4.KafkaPullSpout 配置 Kafka 连接信息，从对应的 Kafka 消息队列中获取数据帧

```
public class KafkaPullSpout extends KafkaSpout {
    private static final long serialVersionUID = -998134271239656863L;
    public KafkaPullSpout(String topic, String task, int maxArraySize, boolean isFromStart)
    {
        super(new SpoutConfig(new ZkHosts(
    "bigdata-w-001.bigdata.com:2181,bigdata-w-002.bigdata.com:2181,bigdata-w-003.bigdata.com
:2181,bigdata-w-004.bigdata.com:2181,bigdata-w-005.bigdata.com:2181,bigdata-w-006.bigdata.co
m:2181,bigdata-w-007.bigdata.com:2181"),
                topic, "/storm_pack", task + "-" + topic, isFromStart));
        this.setMaxArraySize(maxArraySize);
    }
}
```

5.Strom 中启动两个 Bolt，一个对获取的数据帧按规约进行解析，另一个 Bolt 对解析到的数据进行分片，同时将毫秒和时间片数据存储至 Redis。此处不再详细

介绍代码实现。

6.Redis 中数据按约定周期定时转储至 HBASE，并清除 Redis 中对应的数据。核心代码如下：

```
public void HBaseTimer() {
        JedisPool pool = null;
        Jedis jedis = null;
        try {
            pool = RedisAPI.getPool();
            jedis = pool.getResource();
            long currentTime = System.currentTimeMillis() - 5 * 60 * 1000;
            jedis.select(0);
            Set<String> keys = jedis.keys("*|*|*");
            Iterator<String> it = keys.iterator();
            while (it.hasNext()) {
                String key = it.next();
                String[] ss = key.split("\\|");
                long time = Long.valueOf(ss[2]);
                if (time < currentTime) {
                    List<String> list = jedis.lrange(key, 0, -1);
                    String values = "";
                    for (String s : list) {
                        try {
                            if (StringUtils.isNotBlank(s)){
                                values = values + s + "$";
                            }
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                    // 保存到 HBase，删除 Redis 中的 key
                    if (StringUtils.isNotBlank(values)) {
                        Map<String, String> map = new HashMap<String, String>();
                        map.put("rowkey", key);
                        map.put("signalNo", ss[1]);
                        map.put("value", values);
                        HBaseConnectionPool.getInstance().addPackInfo("packinfo", map);
                        jedis.del(key);
                    }
                }
            }
        } catch (Exception e) {
            // 释放 Redis 对象
            pool.returnBrokenResource(jedis);
            e.printStackTrace();
        } finally {
            RedisAPI.getInstance().returnResourceJedis(pool, jedis);
        }

        }
```

5.8 系统测试

大数据基础平台设计为集成和存储企业全类型数据，是企业 IT 架构的基础组件，其面向信息系统，与常规的面向终端用户信息系统不同。大数据基础平台主要需求为满足各系统数据接入和各类数据接入后的平台的存取性能，由企业 IT 运维人员分配大数据基础平台的计算能力、存储能力，为信息系统提供计算功能和存储功能等，因此，作为企业 IT 架构基础组件的大数据基础平台，其系统测试以性能测试和调度测试为主。

5.8.1 数据仓库平台测试

数据仓库平台作为大数据基础平台结构化数据的存储区域，主要支撑后续的业务场景监测分析以及数据挖掘。本次采用 TPC-H 基准测试作为其性能测试依据。TPC-H 是一款面向商品零售业的决策支持系统测试基准，TPC-H 定义了 8 张表，22 个查询，遵循 SQL92。

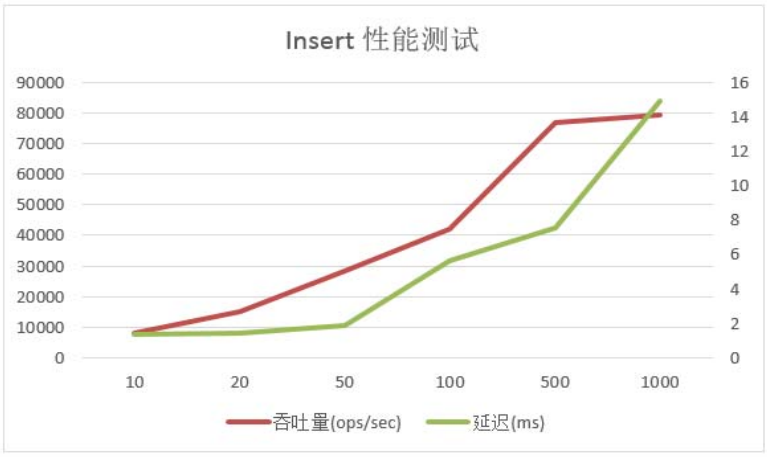
测试结果满足大数据基础平台系统设计，测试结果如图 5-23 所示：

Query 1 finished OK (10 seconds)↵
Query 2 finished OK (2 seconds)↵
Query 3 finished OK (4 seconds)↵
Query 4 finished OK (1 seconds)↵
Query 5 finished OK (1 seconds)↵
Query 6 finished OK (1 seconds)↵
Query 7 finished OK (6 seconds)↵
Query 8 finished OK (4 seconds)↵
Query 9 finished OK (18 seconds)↵
Query 10 finished OK (1 seconds)↵
Query 11 finished OK (1 seconds)↵
Query 12 finished OK (1 seconds)↵
Query 13 finished OK (3 seconds)↵
Query 14 finished OK (1 seconds)↵
Query 15 finished OK (1 seconds)↵
Query 16 finished OK (2 seconds)↵
Query 17 finished OK (8 seconds)↵
Query 18 finished OK (10 seconds)↵
Query 19 finished OK (2 seconds)↵
Query 20 finished OK (1 seconds)↵
Query 21 finished OK (8 seconds)↵
Query 22 finished OK (2 seconds)↵

图 5-23 数据仓库性能测试情况

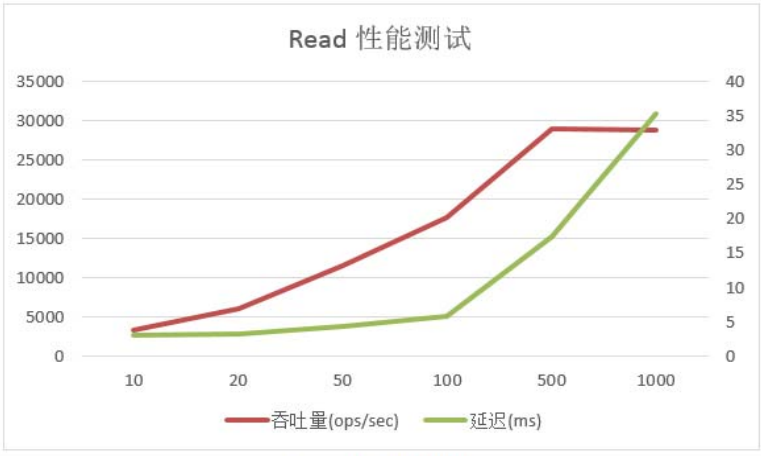
5.8.2 分布式数据平台测试

分布式数据平台作为大数据基础平台非结构化数据和流数据长期存储的区域。根据数据集成需求分析，非结构化的数量量远小于可采集的海量实时数据量。分布式数据平台的性能主要服务于转储的海量实时数据。本测试工具采用 Jmeter，模拟的存储数据大小设定为 64 字节，模拟多线程对系统的插入，读取，查询性能进行了测试，测试性能可完全满足企业大数据平台设计所需，测试结果如图 5-24、图 5-25 和图 5-26 所示：



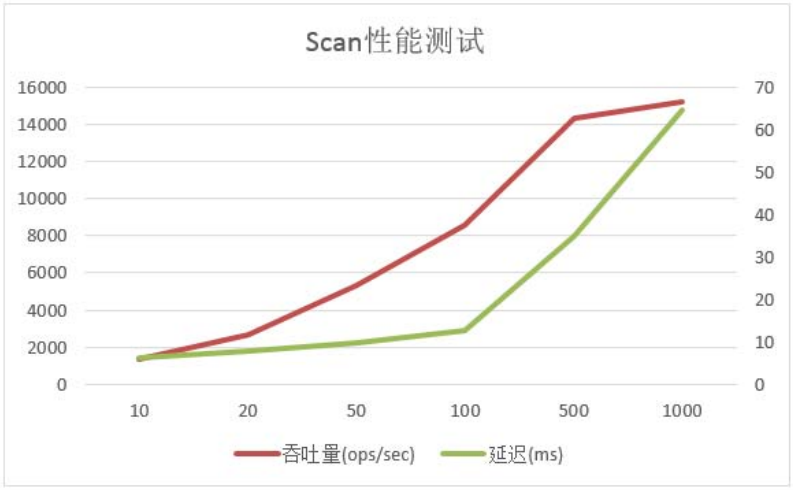
Insert性能测试						
线程数	10	20	50	100	500	1000
吞吐量(ops/sec)	7938	14981	28403	41896	76732	79474
延迟(ms)	1.36	1.46	1.85	5.67	7.55	14.91

图 5-24 分布式数据平台 Insert 性能测试情况



Read性能测试						
线程数	10	20	50	100	500	1000
吞吐量(ops/sec)	3273	6048	11537	17610	28945	28759
延迟(ms)	3.1	3.26	4.35	5.79	17.33	35.2

图 5-25 分布式数据平台 Read 性能测试情况



Scan性能测试						
线程数	10	20	50	100	500	1000
吞吐量(ops/sec)	1360	2701	5333	8593	14305	15220
延迟(ms)	6.43	7.94	9.84	12.71	34.82	64.77

图 5-26 分布式数据平台 Scan 性能测试情况

5.8.3 流数据平台测试

流数据平台需针对海量实时进行高性能存取，根据存储数据的特点开展流数据平台性能测试。根据 Redis 内存数据库单线程的处理机制，利用 Jmeter 测试工具，模拟单线程新增插入记录，数据大小设定 64 字节，进行了 4 中场景性能测试，测试结果能完全满足大数据基础平台设计，测试场景分别为：

- 场景 1：插入 1 千万条固定的 key 值记录，测试结果见表
- 场景 2：在场景 1 的基础上，查询的随机值，执行 1 千万次。
- 场景 3：插入 1 千万条随机 key 值记录
- 场景 4：在场景 3 的基础上，查询的随机值，执行 1 千万次。

测试结果如表 5-3 所示：

表 5-3 流数据平台测试情况

场景 1			
测试次数	第一次	第二次	第三次
耗时（s）	887.03916	926.11008	903.81984
内存消耗（MB）	655.7	655.7	655.7

场景 2			
测试次数	第一次	第二次	第三次
耗时（s）	1014.23616	903.23352	860.3658
场景 3			
测试次数	第一次	第二次	第三次
耗时（s）	1146.957	1147.63656	1198.97148
内存消耗（MB）	744.3	744.3	744.3
场景 4			
测试次数	第一次	第二次	第三次
耗时（s）	1219.9908	1081.22532	1182.85944

5.8.4 ETL 作业调度测试

对 ETL 的作业调度进行测试，登录服务器操作系统，输入命令 crontab -e 配置 ETL 作业调度，crontab 的配置信息如下：

1 1-23/1 * * * sh /home/kettle/job_odm_to_fdm.sh

3 3-23/1 * * * sh /home/kettle/job_fdm_to_adm.sh

5 5-23/1 * * * sh /home/kettle/job_adm_to_mdm.sh

查询数据仓库中定义的作业日志表，可以看到作业执行情况，如表 5-4 所示：

表 5-4 ETL 作业测试执行情况

kjb_name	begin_dttm	end_dttm	status
job_odm_to_fdm	2017-12-13 01:01:00	2017-12-13 02:37:28	Finish
job_fdm_to_adm	2017-12-13 03:03:00	2017-12-13 04:26:31	Finish
job_adm_to_mdm	2017-12-13 05:05:00	2017-12-13 05:47:34	Finish
job_odm_to_fdm	2017-12-13 03:01:00	2017-12-13 03:01:30	Finish
job_fdm_to_adm	2017-12-13 05:03:00	2017-12-13 05:03:30	Finish
job_adm_to_mdm	2017-12-13 06:05:00	2017-12-13 06:05:30	Finish
…	…	…	…

通过作业日志表可以看出 job_odm_to_fdm 作业 2017-12-13 01:01:00 启动，2017-12-13 02:37:28 停止。根据作业调度配置策略，该作业本应在 2017-12-13

02:01:00 再次启动，由于此时上次作业未执行完毕，故本次作业不启动。直到 2017-12-13 03:01:00 时，该作业再次启动。可以确定作业定时调度成功，且防止重复执行功能正常。

查询数据表日志表，可以看到数据表转换执行情况，如表 5-5 所示：

表 5-5 数据表转换执行情况

table_name	begin_dttm	end_dttm	status	row_count
t03_duty_log	2017-12-13 01:03:21	2017-12-13 01:03:22	Finish	317
t03_fire_ticket	2017-12-13 01:03:23	2017-12-13 01:03:25	Finish	4316
t03_work_ticket	2017-12-13 01:03:25	2017-12-13 01:03:26	Finish	1962
…	…	…	…	

通过数据表日志表可以看出，整合区的作业表在 job_odm_to_fdm 作业 2017-12-13 01:01:00 启动后就得到了更新，后续该作业在 2017-12-13 03:01:00 再次启动，以上数据表未重复更新。可以确定防止数据表重复更新功能正常。

5.9 本章小结

在本章中，依照大数据基础平台的设计，完成了大数据基础平台的实现与测试。主要为大数据基础平台组件安装与配置工作和数据集成的具体实现，包括结构化数据、非结构化数据、实时数据采集与存储的具体实现，并对大数据基础平台的数据仓库、分布式数据平台、流数据平台的存取性能以及 ETL 作业调度机制进行了测试。

第六章 全文总结与展望

6.1 本文所做的主要工作论文工作总结

本文采用大数据技术构建了混合架构的大数据基础平台，解决传统架构存在的，扩展性较差，支持数据类型单一，建设成本极高，数据处理能力有限等缺点。解决了企业内部各信息系统的数据孤岛情况，实现了各系统数据的基于主题的集中存储，企业数据自动的持续的接入大数据基础平台，并完成了数据轻度汇总，构建了数据集市，为后续的数据共享，数据驱动企业打下基础。

6.2 进一步的展望

1.随着未来数据的不断接入大数据基础平台，结构化数据会越加庞大，数据仓库在存储和使用海量结构化数据上已然会都存在瓶颈，本次大数据基础平台可进一步扩展。在分布式平台上扩展归档保存海量的结构化数据；在大数据分析时，利用 ETL 将数据读取到分布式数据平台，利用分布式计算的优势处理数据并返回结果。

2.进一步建设规范化和标准化的数据共享服务，通过创建数据共享区，开放数据查询接口等方式，提供各系统对各类数据的订阅、访问，实现大数据基础平台的数据对外共享。

3.进一步基于大数据基础平台，开展数据的利用和展示。利用数据筛选、变量转换已经长时间序列信号数据的时频域分析等数据预处理功能，利用如聚类、分类、回归分析、关联分析和时序预测等数据挖掘算法，运用各种图表完成对企业核心业务、关键流程等关键指标的直观展示，对企业关键数据穿透关联，实现核心指标的实时动态监控，充分发掘数据价值。

4. 大数据基础平台所包含的组件众多，功能复杂，目前各组件的配置均在其各自配置页面进行，组件监控管理也是其各自监控管理页面，后续可以建立一个独立的组件管理平台，在平台上直接可以配置、监控大数据基础平台的各个组件。

致　谢

此论文的完成，我要衷心的感谢我的导师郭建东。郭老师的深厚学术知识，科学严谨的态度，让我受益匪浅。在我工程硕士论文的选题至撰写的全过程中，XXX 给予了我无私和深切的指导，协助我在软件工程专业知识上的学习，指正论文中的不足之处，才使我能顺利完成了此篇工程硕士论文。

在电子科技大学攻读工程硕士已经有三年时间，电子科技大学给我们提供了坚实的学习平台，经过 3 年的的学习，让我获益良多，对自身素养和岗位工作都有极大的帮助，感谢电子科技大学所有老师和同学。

感谢在我答辩期间为我指导的各位老师，是你们的专业素养才使我不断前进。谢谢老师们的指教，我将积极接受你们的指正和教导。

最后我还要感谢我的家人，是他们默默的支持才让我可以完成工程硕士的学习和研究。

参考文献

[1] 严霄凤,张德馨. 大数据研究[J]. 计算机技术与发展,2013,23(4):168-172.

[2] 曲建峰,孙 翌,徐汝兴等.Oracle RAC 集群技术在图书馆集成管理系统中的应用[J]. 现代图书情报技术,2012(7/8):133-138.

[3] 杨锐. 大数据环境下动态数据仓库的应用研究[J]. 电子技术与软件工程, 2015(2):215-215.

[4] 肖波,景帅,吴建军等.基于模型驱动的中国石化企业数据中心模型架构[J]. 大庆石油学院学报,2012，36(1):,78-81.

[5] 王山水. 海量数据离线存储相关实施标准探讨(续前)[J]. 数字与缩微影像, 2014(4):20-23.

[6] 李国杰,程学旗.大数据研究:未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考[J].中国科学院院刊,2012,27(06):647-657.

[7] 杨德胜,陈江江,张明.电力大数据高速存储及检索关键技术研究与应用[J].电子测试,2014(03):62-63+61.

[8] 杨正益. 制造物联海量实时数据处理方法研究[D].重庆大学,2012.

[9] 王德文,宋亚奇,朱永利.基于云计算的智能电网信息平台[J].电力系统自动化,2010,34(22):7-12.

[10]衡星辰,周力.分布式技术在电力大数据高性能处理中的应用[J].电力信息与通信技术,2013,11(09):40-43.

[11]冯歆尧,彭泽武.基于广东电网数据仓库大数据架构研究[J].电力学报,2015,30(06):519-523.

[12] 孟祥萍,周来,王晖,纪秀.HBase 在智能电网异构数据同步中的应用[J].电力系统保护与控制,2015,43(24):122-128.

[13]崔有文,周金海.基于 KETTLE 的数据集成研究[J].计算机技术与发展,2015,25(04):153-157.

[14]Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. Acm Sigops Operating Systems Review, 2003, 37(5):29-43.

[15]White T, Cutting D. Hadoop : the definitive guide[J]. O'reilly Media Inc Gravenstein Highway North, 2009, 215(11):1 - 4.

[16]Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[M]. ACM, 2008, 1(2):55-56.

[17]Qin X P, Wang H J, Xiao-Yong D U, et al. Big Data Analysis—Competition and Symbiosis of RDBMS and MapReduce[J]. Journal of Software, 2012, 23(1):32-45.

[18]Harter T, Borthakur D, Dong S, et al. Analysis of HDFS under HBase: a facebook messages case study[M]. USENIX Association, 2014.

[19] Stonebraker M. SQL databases v. NoSQL databases[J]. Communications of the Acm, 2010, 53(4):10-11.

[20] George L. HBase : the definitive guide[J]. Andre, 2011, 12(1):1 - 4.

[21] Aiyer A, Bautin M, Chen G J, et al. Storage Infrastructure Behind Facebook Messages Using HBase at Scale[J]. Bulletin of the Technical Committee on Data Engineering, 2013, 35(2):000996-000999.

[22] Lassnig M, Garonne V, Dimitrov G, et al. ATLAS Data Management Accounting with Hadoop Pig and HBase[C]// 2012:2044.

[23] Ting K, Cecho J J. Apache Sqoop Cookbook[M]. O'Reilly Media, 2013.

[24] 于金良,朱志祥,梁小江.一种基于 Sqoop 的数据交换系统[J].物联网技术,2016,6(03):35-37.

[25] Yang W, Liu X, Zhang L, et al. Big Data Real-Time Processing Based on Storm[J]. 2013, 8(1):1784-1787.

[26] 靳永超,吴怀谷.基于 Storm 和 Hadoop 的大数据处理架构的研究[J].现代计算机(专业版),2015(04):9-12.

[27] 王震,陈亮.基于 Kafka 消息队列的电网设备准实时数据接入方法研究[J].山东电力技术,2015,42(06):41-43.

[28] Wang G, Koshy J, Subramanian S, et al. Building a replicated logging system with Apache Kafka[J]. Proceedings of the Vldb Endowment, 2015, 8(12):1654-1655.

[29] 卢冬海,何先波.浅析 NoSQL 数据库[J].中国西部科技,2011,10(02):15-16+14.

[30] Tinnefeld C, Zeier A, Plattner H. Cache-conscious data placement in an in-memory key-value store[C]// Symposium on International Database Engineering & Applications. ACM, 2011:134-142.

[31] Berk Atikoglu,Yuehai Xu,Eitan Frachtenberg,Song Jiang,Mike Paleczny. Workload analysis of a large-scale key-value store[J]. ACM SIGMETRICS Performance Evaluation Review,2012,40(1).

[32] 袁晓洁, 孙国荣. 数据库原理和实践教程[专著] : Gbase 8t Based on Informix 剖析与应用[M]. 电子工业出版社, 2016.

[33] 林烈青. 企业数据中心的研究与设计[J]. 制造业自动化. 2011,33(8):22-25.

[34] 李晓芳. 数据仓库技术在水电调度管理系统信息集成中的应用研究[D]. 河海大学, 2004.

[35] 朱少敏,刘建明,刘冬梅.基于模糊神经网络的电力企业数据中心绿色评价方法[J]. 电网技术,2008，32(19):,84-88.