密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：基于 Hadoop 的海量业务数据分析平台的设计与实现

学位类别：专业学位

学　　号：2011140677

姓　　名：魏迪

专业领域：计算机技术

导　　师：王洪波

学　　院：网络技术研究院

2013 年 12 月 28 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

基于 Hadoop 的海量业务数据分析平台的设计与实现

摘 要

随着互联网技术的进步，信息社会不断发展，数据呈爆炸式的增长。一方面数据量的增长给企业的存储要求和计算要求带来了新的挑战，另一方面数据中蕴藏着巨大的有用信息，企业的运营需求也在增加。对于企业来说这些数据中包含着用户使用产品的特征，用户对于哪种产品更加喜好以及用户的行为特征等等信息，这都是企业掌握市场动向的数据来源。然而传统的数据分析平台中，数据一般存储在关系型数据库中。但是数据量的不断增长，单服务器执行任务的这种特性，不管是在计算还是存储能力上都遇到了瓶颈，在其扩展性上也有很大的限制。因此，本课题设计和实现了基于 Hadoop 的分布式架构的海量业务数据分析系统。

本文首先介绍了课题的相关背景，研究了与本课题相关的技术理论知识，包括 Hadoop 整体框架，Hive、Flume、Redis 开源组件。随后结合企业人员的运营分析需求对本系统进行了详细的需求分析，包括功能性需求，非功能性需求以及用例分析。接着对系统进行了整体架构设计，将系统分为五个设计模块，数据收集模块、数据预处理模块、数据分析模块、监控模块、数据展示模块。根据各个模块的具体需求，本文对各个模块进行了详细设计，结合流程图和相关代码进行了具体描述。同时，在数据各个环节中进行了监控，保证数据的准确。最后对系统进行了数据测试和效果展示，表明达到了预期效果。

本课题设计的海量业务数据分析系统基于分布式架构，克服了传统数据库计算效率低、存储空间小的缺点，将大的数据文件分散到各个数据节点进行并行计算，分散了计算负担，提高了计算效率。同时，本系统提供了良好的数据备份和错误恢复能力。目前本系统已经在某公司正式应用。

关键词：数据分析 分布式 Hadoop Hive Flume

DESIGN AND IMPLEMENTATION OF SERVICE DATA ANALYSIS SUBSYSTEM BASED ON HADOOP

ABSTRACT

The rapid development of the Internet and information society has brought a huge amount of data. Data growth on one hand brings the higher requirements of calculation and storage, on the other hand there is great value in it. So the demand of business operations is also increasing. The user behavior characteristics and product usage contained in these data is an important source of information for enterprises to grasp market trends. However, for the traditional data analysis platform, the data is stored in a relational database. The computing speed and storage capacity and scalability are greatly limited due to the single-server to perform tasks and the growing data. Therefore, this thesis designed and implemented the data analysis system based on Hadoop distributed architecture.

At first, this thesis introduces the background of the subject and theoretical knowledge associated with this issue which includes Hadoop, Hive, Flume and Redis. And then according to the user requirements, this thesis makes the functional and non-functional requirements analysis with use cases diagram. Then this thesis divide the system into five modules, including data collection module, data preprocessing module, data processing module, monitor module and data representation module. Based on the requirements of each modul, this thesis makes a detailed introduction about the specific design and implementation of each module combined with sequent diagram and code. Meanwhile, in all aspects of the data flow are monitored to ensure the consistency of the data. Finally, the system is tested and shows the results, which indicates that it has achieved our goals.

The data analysis system based on the distributed architecture is able to overcome the shortcomings of traditional databases, including slow

calculation and small storage space. It can split the large files into pieces on the distributed file system server, dispersing the stress of server and improving the the efficiency of calculation.At the same time,this system provides good backups and error recovery mechanisms. At present the data analysis system has been used in a company.

KEY WORDS: Data Analysis, Distributed, Hadoop, Hive, Flume

目 录

第一章 绪论

1.1 研究背景及意义

信息化是当今社会发展的主要方式，信息化的发展使得数据生成的速度不断加快，数据呈爆炸式的增长。大数据[1]的提出推动了 IT 产业技术的发展，成为了又一次技术变革。大数据是指所处理的数据量巨大，普通的数据分析软件无法在短期时间内对数据进行处理完毕。大数据对数据的存储和数据的处理方法提出了新的要求。科技发展速度不断加快，人们对于互联网产生越来越大的依赖。人们上网联系朋友、购物、分享心得，发表意见、浏览新闻等，随之产生数以 TB 甚至 PB 级别的数据。在这种情况下，我们如何在这些海量的数据中分析出对于企业有用的信息成为关键。近些年来 Hadoop 已经逐渐成为海量数据处理和存储的主要方式。Hadoop 分布式计算平台提供了高可扩展性、高可靠性、高效性以及高容错性，使其迅速成为各大企业的主要数据分析平台。有研究报告称，到 2015 年，大约 65%的具有高级分析功能的打包分析应用将嵌入 Hadoop。

随着科技手段的不断进步，企业也逐渐的发现数据价值越来越大，然而这些数据不能直观的体现出需求，需要对数据进行加工分析才能呈现。数据分析就是从海量的业务数据中提取出对企业有用的信息。比如，分析用户使用软件的情况，软件的安装、卸载情况，这些信息可以知道产品的受欢迎程度，用户的喜好，用户的行为特征等等。这对企业在进行市场推广有很大的价值。在数据分析领域做得好的企业甚至向商家推出了数据分析的服务，让数据分析直接为公司带来了收入。

Hadoop 分布式框架的出现推动了分布式计算的发展，Hadoop 可扩展性是一大优势。Hadoop 设计目标是实现移动计算和存储的可扩展。Hadoop 对硬件的要求不高，可以运行在任何的普通电脑机器上。Hadoop 数据可靠性好，Hadoop 存储是数据冗余存储，每份数据都保存多份的备份，保证数据的安全性。Hadoop 处理数据高效，MapReduce 任务是分散在多个节点并行执行的，并且实现了程序移动数据不动的移动计算方式，减少了数据传输带宽，提高数据处理效率。

1.2 国内外相关研究

随着越来越多的数据信息，如何从海量的数据中提取出有效的信息成为如今的一个重要课题。近年来，数据挖掘引起了各个产业界非常高的关注，并迫切的需要从这些海量的数据中提炼出有用的信息和知识。简单的说，数据挖掘[2]就是

从海量的数据中提取或“挖掘”知识，它是提取准确、新颖、有用并易于理解的数据处理过程。它与常用术语数据库中的知识发现（Knowledge Discover Database，KDD）[3]有着密切的联系。KDD 是一个多环节处理过程，分为：数据清理，数据集成，数据选择，数据变换，数据挖掘，模式评估，知识表示几个步骤。

随着技术的不断更新，数据分析技术也在与时俱进，从关系型数据库到数据仓库，再到现今的分布式计算时代。

关系型数据库系统（DBMS）[4]，由一组相关的内部数据和一组管理和存取这些数据的软件程序组成，采用关系型模型来组织数据库。关系模型是由 IBM 研究员 Edgar Frank Codd 1970 年首次提出来的。1974 年，IBM 的 Don Chamberlin 和 Ray Boyce 根据“科德十二定律”[5]描述出了 SQL[6]（Structured Query Language）语言。随着 IBM SystemR 的出现，说明了在商业环境中一个全功能关系数据库是完全可以使用的。随着 IBM 发布了数据库产品 DB2，不久，Oracle 公司也紧跟着发布了数据库产品 Oracle，Sybase 和 Informix 也加入了关系型数据库竞争行列。这些标志着数据分析进入一个新的时代。关系型数据库都是以 SQL 语言标准为操作语言，架构体系使用实体关系模型，提供了一些常用的关系操作，例如查找、关联、并、差、交等操作，也提供了函数、存储过程等结构化语言。同时关系型数据库保证了数据实体完整性、参照完整性、用户定义完整性。当数据量较小时候，关系数据库很好的满足了用户数据分析、数据计算的需求。

随着数据量的增加，数据开始多元化。数据仓库[7]满足了分析人员对数据多元化的要求。数据仓库是一个面向主题的（Subject Oriented）、集成的（Integrate）、相对稳定的（Non-Volatile）、反映历史变化（Time Variant）的数据集合，用于支持管理决策[8]。数据仓库把数据以维度和数据项的形式展示出来，数据的维度就是用户观察数据的角度，是用户分析数据的出发点，数据项观测出来的数据的值。数据仓库这种组织数据的方式，一方面可以体现出数据的立体特性，另一方面人们获取某一维度的值更加直观方便。数据仓库通常的数据分析方式是按照周期进行分析，最常用的是以年月日的方式，日报是最常见的，通过日报分析可以看出用户发展趋势，用户生命周期等等。

随着数据量的不断增大，数据仓库的计算延时慢慢体现出来，有的企业数据仓库架构非常复杂庞大，可扩展性不强，严重影响了未来运营的发展，数据量的剧增也使得数据仓库不再具备复杂数据分析的条件。为了解决上述计算效率，可扩展性等一系列的问题，分布式计算被应用于数据分析的工作中。

分布式计算是把一个任务分配到多个机器上去并行执行，大大提高了计算效率。同时分布式系统对服务器要求不高，有较大的存储空间，并且实现了负载均衡。

分布式系统的发展离不开 Google 项目的推动，Google 推出了 Google fs [9]，MapReduce [10]，Bigtable [11]影响计算格局的三个产品，虽然没有开放源代码，但是发布了产品设计的三个论文。Apache 的 Hadoop 项目就是根据 Google 的论文进行了具体实现，Hadoop 的 MapReduce 模型和谷歌的 MapReduce 模型是一个原理，Hadoop Distributed File System（HDFS）和 Google fs 对应，Hbase 和 Bigtable 对应。

Hadoop 的优势一是它可以进行扩展，Hadoop 的设计思想是实现存储设备的可扩展和计算能力的可扩展，对 Hadoop 进行扩展是非常容易的，不需要修改任何已经存在的结构。二是 Hadoop 的经济实用，Hadoop 框架的运行不需要是昂贵的服务器，它可以运行在任何普通的 PC 机上，对系统硬件没有特殊的要求。三是集群的可靠，HDFS 分布式文件系统提供备份恢复机制和 MapReduce 的任务监控功能，这些都使得数据处理的安全可靠得到保证，同时还具备负载均衡和异常恢复机制，为高效处理海量的数据提供了安全保障。四是 Hadoop 处理数据的高效性，一个任务分派到多个节点同时进行处理，减少了单台服务器的负载，提高了整体计算效率。

1.3 本文研究内容

本课题主要任务是利用 Hadoop 搭建分布式计算环境，并且在该环境下部署数据分析任务。研究 Hadoop 系统的架构体系，对其分布式文件系统 HDFS 和 MapReduce 并行编程模型的原理和实现进行深入理解。要想利用 Hadoop 分布式环境处理数据，就要要求用户自己开发 MapReduce 程序，而这种 MapReduce 程序处于比较底层的层次，用户不容易方便操作及掌握，而且也很难维护。即使一个简单的查询任务用户也得花费大量的时间去编写 MapReduce 程序。这种情况下，想要发挥最大化的海量数据处理任务就比较困难了。怎么降低 Hadoop 程序的开发难度，就显得尤为重要了。

Hive 是一个开源的数据仓库工具，它是基于 Hadoop 架构的。它引入了传统数据库中的表、分区、SQL 等概念，使得 Hadoop 开发和普通写数据库 SQL 语言一样简单，同时也保留了 Hadoop 的可扩展性。

本文将基于 Hadoop 和 Hive 架构实现一个大数据的分析平台系统。主要处理以下几个问题：

1. 数据收集，主要针对不同类型不同服务器的数据进行数据的集合。包括数据库数据，日志数据等，保证数据在传输过程中的完整性。
2. 数据的预处理，主要包括数据清洗，数据集成，用户数据初始化。
3. 数据分析，根据用户具体需求，对计算请求迅速响应，支持各种数据的

处理，计算。

4. 数据展示，主要是对数据处理结果进行展示，使得数据更直观、更便捷的展示给用户。
5. 监控告警，主要针对数据处理过程中的异常情况进行监控告警，以及服务器的监控。

本论文主要是为了解决上述问题，设计出可行的技术方案并实现，最后在实际企业中进行测试和部署使用。

1.4 论文组织结构

本文共分为六章，各章节的主要内容如下：

第一章：绪论，介绍了本文的研究背景，国内外相关问题的研究，本文的主要研究内容以及论文的结构。

第二章：相关技术背景介绍，主要研究了 Hadoop 基础框架，包括 HDFS 分布式文件系统和 MapReduce 编程模型，Hive 组件工具的研究，Flume 数据传输的研究以及 Redis 技术的优势。

第三章：需求分析及架构设计，主要介绍了数据分析平台系统的需求以及各个模块的设计架构。

第四章：系统的详细设计及实现，主要介绍了数据分析平台系统的各个模块的具体设计以及实现方法。

第五章：系统部署与测试，主要介绍了系统的硬件环境，数据分析的验证，以及数据结果的展示。

第六章：总结和展望，主要对论文进行总结，提出系统中的优势和不足，并对未来的发展进行展望。

第二章 相关技术研究

本文的设计和实现是基于 Hadoop 分布式架构和 Hive 架构的，利用 Hadoop 并行处理数据的能力对大量数据进行分析。本章着重介绍 Hadoop 体系的整体架构，包括 HDFS 分布式文件系统和 MapReduce 模型，还介绍了 Hive 语言语法，Flume 数据传输以及 Redis NoSql 数据库的应用优势。

2.1 Hadoop 介绍

Hadoop[12,13]是由 Apache 基金开发的一个开源框架的项目，提供了分布式的、可扩展、可靠的分布式处理功能。Hadoop 框架上可以编写和执行分布式应用程序用来处理大规模的数据。他不仅仅支持单机模式也支持成百上千台服务器集群的模式。Hadoop 最先被提出是因为 Google 针对大规模数据而推出的 MapReduce 技术。

Hadoop 对机器的配置要求并不高，认为在处理任务时失败是合理的，所以 Hadoop 高性能并不依赖于硬件的资源。Hadoop 同样允许数据在一定范围内丢失，Hadoop 有足够大的存储空间，所以 Hadoop 保存的是冗余数据，每份数据都有多个备份，一份数据丢失后备份数据会及时恢复，保证数据一直存在 n 份。

Hadoop 分布式架构比其他的分布式系统有许多的优点。 首先，Hadoop 是由 Java 语言开发实现的，Java 语言具有非常高的可移植性，因此 Hadoop 可以部署在更广泛的机器上。其次，Hadoop 是一个开源的框架，所以，有很好的可扩展性。Hadoop 不依赖于硬件，成本低，可以使用很普通的服务器甚至 PC 机组成集群处理大规模数据，服务器集群节点可以达到上千个。最后，Hadoop 计算效率高，一个任务同时有多个服务器进行计算，各个服务器之间并行计算互不影响，大大缩减了数据计算的时间。

Hadoop 的处理任务主要是针对数据比较密集的情况下，一般这种任务的特点就是数据量特别的大，数据在各个计算环节中的传输就变的十分困难。而 Hadoop 集群既可以存储数据又可以作为数据处理的平台。客户端只需要向 Hadoop 发布所需要执行的 MapReduce 程序即可，使得执行任务转移到数据存储的机器上，在整个数据运算的过程中，数据是保持不动的，而是让处理程序进行迁移，这种设计保证了客户端不会和集群上的源数据进行交互传输，极大减少了数据在传输过程中所占的带宽，充分利用了每个节点的计算能力，同时又降低了 IO 压力，避免了数据计算时的传输问题。

Hadoop 生态系统中包含很多开源组件，其中最重要的是 MapReduce 和 HDFS。

MapReduce 的核心概念就是任务的分配与结果的汇总。HDFS 是 Hadoop 分布式文件系统的简称，是分布式计算中数据存储的基础，是 Hadoop 的核心技术。下图是 Hadoop 生态系统的一些常用组件。

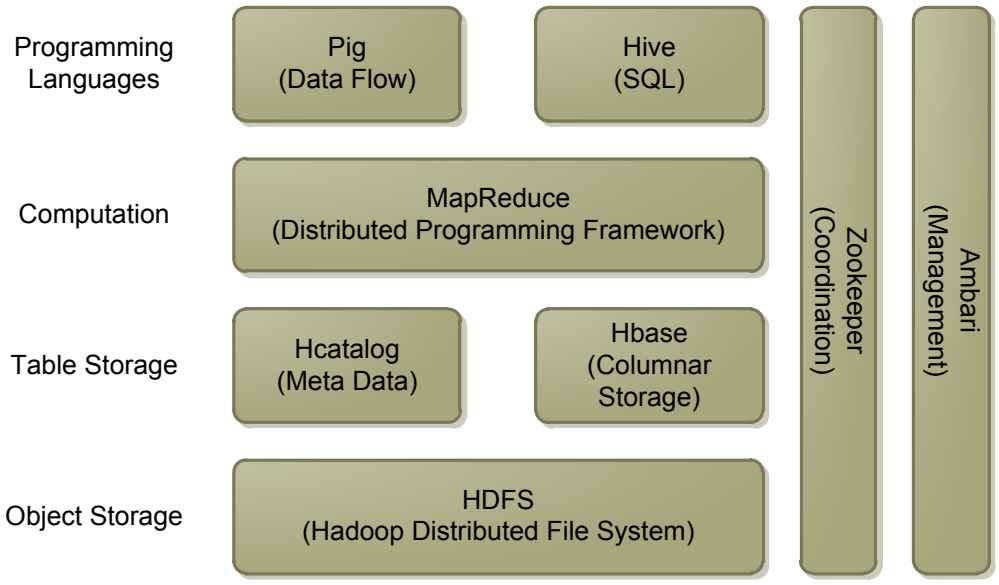

图 2-1 Hadoop 生态系统

2.1.1 HDFS

HDFS（Hadoop Distribution File System）[14,15]是 Hadoop 项目的核心基础子项目，是 Hadoop 框架的分布式文件系统，为 Hadoop 中的其他应用提供可靠的数据存储和高效的数据访问。HDFS 是 Hadoop 架构中最底层的文件系统，是其他项目实现的基础。

HDFS 架构由一个 Namenode 主控节点和多个 Datanode 从节点构成。如下图：

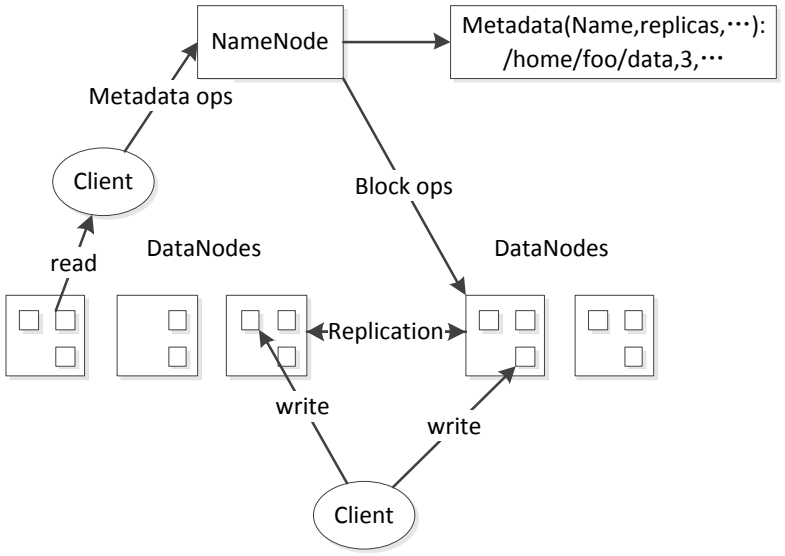

图 2-2 HDFS 架构图

Namenode 节点负责保存管理分布式文件系统的元数据，记录文件的每个操

作。NameNode 利用一个被称为 EditLog[16]的事务日志去记录文件系统元数据所发生的每一次改变。Datanode 节点存储实际具体的数据。每个对于数据的操作都要从 NameNode 节点获取数据位置信息然后再从 Datanode 节点读取数据。

HDFS 中文件的存储仍然采用和 Google 的 GFS 的中文件按块存储一样的方式。每个文件分为大小相同的块（Block）。文件存储处理的逻辑单元就是块。块是一个抽象的概念，代表了系统读写数据的过程中可操作的最小的单元。也就是说，文件被分为大小相同的块，放到 HDFS 中，块与块之间没有联系。数据读取的时候只能操作块大小的整数倍数据。每个块的信息储存在一个特定的位置，保存信息的数据被称为“元数据”。HDFS 中每个数据块默认的大小是 64M，也可以通过修改系统参数人为设定大小。这样就可以有效的减少内部碎片，防止空间浪费。为了防止数据的丢失，HDFS 对于每一个块都进行冗余存储，默认每个数据块保存 3 份，也可以进行人为设定。

NameNode 节点一般只有一个，所有的任务都有通过它来获取数据节点的信息。所以 NameNode 节点一般采用大内存、IO 大的服务器，同时它的运行是否正常是整个集群正常工作的关键。为了减少 NameNode 的负载，一般情况下不会在 NameNode 节点保存实际用户数据，也不会在 NameNode 节点执行 MapReduce 任务。NameNode 节点的唯一性也导致了系统潜在的危险，即 Hadoop 集群的单点失效，如果 NameNode 节点出现问题，会影响整个 Hadoop 系统，不能通过简单的重启进程来恢复系统运行。因此，NameNode 节点要使用性能和稳定性较好的服务器，同时，HDFS 系统中还存在一个 SecondaryNameNode 节点，定期的备份 NameNode 节点上所存储的数据块的管理信息，当 NameNode 节点出现故障的时候我们能够快速的将 HDFS 恢复到最近的一个状态。因此，应该将 SecondaryNameNode 节点和 NameNode 部署在不同的机器上，并且要做到定期备份。

DataNode 节点是文件系统的工作节点，它们响应数据块的创建、复制和删除请求，并对本地数据块进行管理。它储存着用户的具体数据，并根据 NameNode 发出的指令执行具体的数据处理的任务，包括写入、读取、删除数据块。

当读写 HDFS 文件时，文件被分割为大小相等的数据块，客户端从 NameNode 节点获取这些数据块所在 DataNode 节点的位置，或者数据应该写入 DataNode 节点的具体物理位置，客户端与 DataNode 节点直接通信，进行文件的读写。

文件读写[17,18]是一个文件系统最基本的功能。当 HDFS 客户端需要从 HDFS 文件系统中读取数据时，客户端需要首先向 NameNode 节点发送一个读文件的请求，NameNode 节点会查询命名空间找到文件所有数据块的信息，返回给客户端这些数据块所在 DataNode 的位置信息，客户端选择相应的数据块读取上面的数

据，全部数据块读取完毕之后，关闭与 HDFS 文件系统的连接。如果在读取过程中，客户端与 DataNode 节点通信出现错误，则客户端会尝试连接包含此数据块的另一个 DataNode 节点，失效信息上报给 NameNode 节点，标记失效数据以及复制数据，保证数据的冗余。

HDFS 文件系统写文件过程要复杂的多，同样客户端先向 NameNode 节点发送一个写文件的请求，NameNode 节点首先检查所要创建的文件是不是已经存在，客户端是不是有权限创建文件，一切都检查通过之后，则创建一个文件，否则返回异常信息。客户端开始写文件时，需要将待写入的文件分成多个数据块，默认情况是每个数据块 64M，也可以通过配置参数修改，数据块保存在数据序列中，然后向 NameNode 申请保存数据块的 Blocks，获取用来保存具体数据的 DataNode 列表，分配的数据节点保存在 pipeline 中。Client 将数据块写入到 pipeline 的第一个节点中，第一个 DataNode 节点将写入数据块发送给 pipeline 中第二个 DataNode 节点，然后依次类推，一直到最后一个 DataNode 完成数据块的写入。最后一个 DataNode 节点写入数据块成功以后会通过 pipeline 返回确认包给客户端，在客户端维护着确认包，成功收到 DataNode 写入结束标志的确认信息后，会从确认序列中移除相应的节点信息。

HDFS 特点：

硬件故障：通常 HDFS 文件系统运行在上百台服务器上，服务器集群中每个节点都存储着文件系统的一部分数据。在整个集群中，任何一台服务器都有可能出现故障，因此在集群中认为硬件故障是属于合理的。故障的检测以及发生故障后的自动恢复是 HDFS 架构设计的首要目标。

大数据存储：HDFS 的设计理念是在 Hadoop 框架下实现大数据的分布式分析处理，同时存储 GB、TB 到 PB 的大数据集。HDFS 把大数据文件按照一定的大小分割为多个数据块保存在文件系统中，同时提供了很高的数据传输带宽。一个 HDFS 集群中节点数可扩展到成百上千个，可以存储数以千万计的文件。

数据流的访问：HDFS 设计用于批处理操作而不是交互式应用。主要是流数据的访问方式，HDFS 相对于数据延迟，更注重于数据的吞吐量[19]。

数据访问模型简单一致性：HDFS 采用一次性写入，多次性读取的文件访问方式。数据一旦被写入文件就不允许再修改，这种方式简化了数据在一致性问题上安全隐患，并且提高了数据访问的吞吐量。

移动计算[20]：HDFS 中存储的数据是非常庞大的，如果一个任务运行时需要把数据都读取出来那是非常消耗时间的。因此，HDFS 把计算的请求放到了数据存储的本地附近执行，这样大大提高了执行效率。有效减小了网络的拥堵。

2.1.2 MapReduce

Hadoop 中的 MapReduce[21,22]并行处理模型是 Hadoop 的核心架构，来源于 Google 发表的论文 MapReduce 技术。MapReduce 可以解决很多大数据环境下的数据集中型作业，并且设计的程序本身也是为了运行在分布式环境上，十分适合并行计算。

MapReduce 是将一个大型的分布式计算表示为一个对数据键值对（Key-value）集合进行串行化并行操作的编程模型。Hadoop MapReduce 架构利用计算机集群，将用户定义的 MapReduce 任务按照 Key 值分散到集群中的节点计算机上进行执行。一个 MapReduce 运算过程包括两个阶段，一个是 Map 阶段另外一个是 Reduce 阶段。计算的输入是一个键值对数据集合，输出的也是一个键值对集合，键值映射的思想贯穿于整个 MapReduce 计算过程。

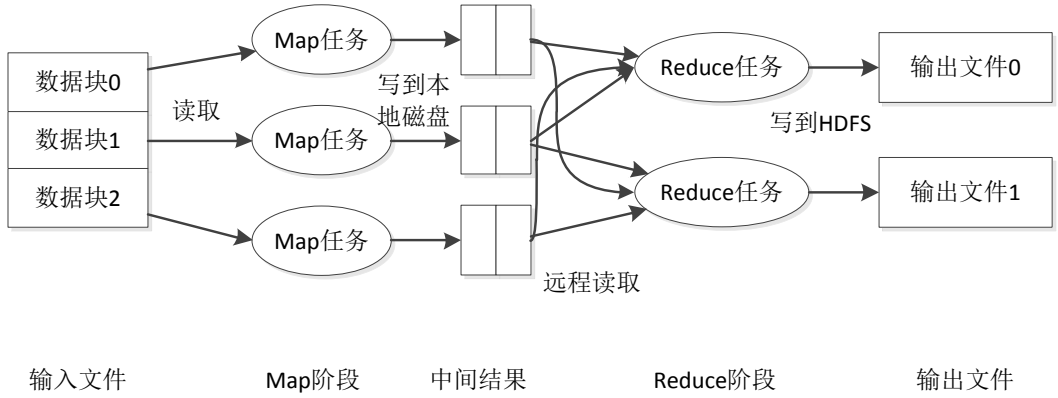

图 2-3 MapReduce 数据处理流程图

用户处理的数据会解析成键值对的形式，初始情况的 Key 为每行数据的偏移量，values 为每行数据。每一个键值对作为 Map 端的输入。根据具体需求，Map 函数中自定义对于数据的处理操作，例如统计文件中各个单词的数量，Map 输入的是文件中每行数据的偏移量和数据内容，Map 函数对数据内容进行分割处理，单词作为 Key 值，单词数量作为 value 值进行输出。这样就生成了新的键值对，这个新的键值对就是 Map 的输出。在单个节点的多个 Map 完成的基础上，输出整合成一个键值对列表，作为下一数据处理环节的数据来源。

Map 函数针对于一条数据进行处理操作，将原始数据进行分割，在多台节点同时进行 Map 操作，实现了数据的并行计算，大大提高了数据处理效率。发挥出分布式计算环境的优势。

Reduce 过程是将 Map 函数的输出结果进行规约处理的过程。分布式 Map 并行处理的过程中并没有逻辑上的先后顺序，因此，Map 函数输出的键值对列表要先经过预处理才能进入 Reduce 函数中。键值对中能够进行排序的是 Key 值部分，预处理过程先按 Key 值进行处理，将各个节点中键值对进行整理，相同 Key 值

的进行合并，Key 值不变，value 变为一个列表，如<Key，list(value)>，新的键值对作为 Reduce 的输入。Reduce 函数针对键值对自定义编写处理过程，通常情况下是操作 Key 值所映射的 value 列表的数据，例如最大值、平均值、求和等操作。一般情况下，Reduce 的输出也是以键值对的形式写入输出文件。

一个日常的 Hadoop 任务作业中，通常包含多个执行 Map 函数的节点，称之为 Mapper。同样，执行 Reduce 函数的节点称之为 Reducer。Mapper 数量表示并行执行的进程数，Mapper 越高表示并行执行程度越强。但并不是说 Mapper 数越多执行效果越高，这个还和数据本身特性以及数据量的大小相关。Reducer 的数据也对任务的执行效率有一定的影响。Reducer 与 Mapper 不同，Mapper 不用关心数据的顺序，可以高效的并行执行，数据的结果也是间断无序的，这些数据要放入 Reducer 中就必须对数据进行整理合并，当然我们也可以让一个 Reducer 来处理，这样就相当于单机处理了，失去了整个数据集群分布式计算的作用。造成了数据处理的瓶颈。在执行 Reduce 之前，每个 Reducer 会对数据进行整理排序，把相同 Key 值的键值对合并，不同键值对分发给不同的 Reducer 处理。

Hadoop 集群的每一次用户计算请求，被称为作业（Job）。Hadoop 就是要把这个任务分发下去，交给具体的任务执行节点来完成这个作业。与 HDFS 中的主从结构类似，在 MapReduce 架构中也采用了 master/slave 这种类似结构，主要分为三类服务器，叫做 JobTracker，TaskTracker 和 JobClient。在 MapReduce 架构中 JobTracker 作为任务的主服务进程即 master，负责管理运行在此框架下所有作业的，作为一个调度中心，它为各个节点分配任务。

TaskTracker 就是 slave，负责执行具体的任务操作。每一个 Job 根据用户程序内容，被分成 Map 和 Reduce 等多项子任务分配到适合的任务服务器上。负责将 Job 提交到 JobTracker 上的一个单元被称为客户端（JobClient），用户要经由客户端相关的代码，将 Job 作业及其相关内容和配置提交到 JobTracker 上去。一个基本的 MapReduce 程序提交 Job 的时序图如图 2-4 所示：

JobTracker 是应用程序和 Hadoop 之间的桥梁，作为一个主服务进程，代码一旦提交，JobTracker 首先会创建一个 JobInProgress 实例，该实例跟踪调度这个 Job，将 Job 加入队列中并依照优先级确定执行计划。JobInProgress 会按照提交的 Job 的配置中定义的输入参数，创建对应的 TaskInProgress 来对 Map 和 Reduce 任务进行监控和调度。与 HDFS 的主控服务 NameNode 类似， JobTracker 一般被部署在单独的机器上，一般 NameNode 和 JobTracker 部署在同一个节点上，这样的设计大大简化了同步操作所带来难度。

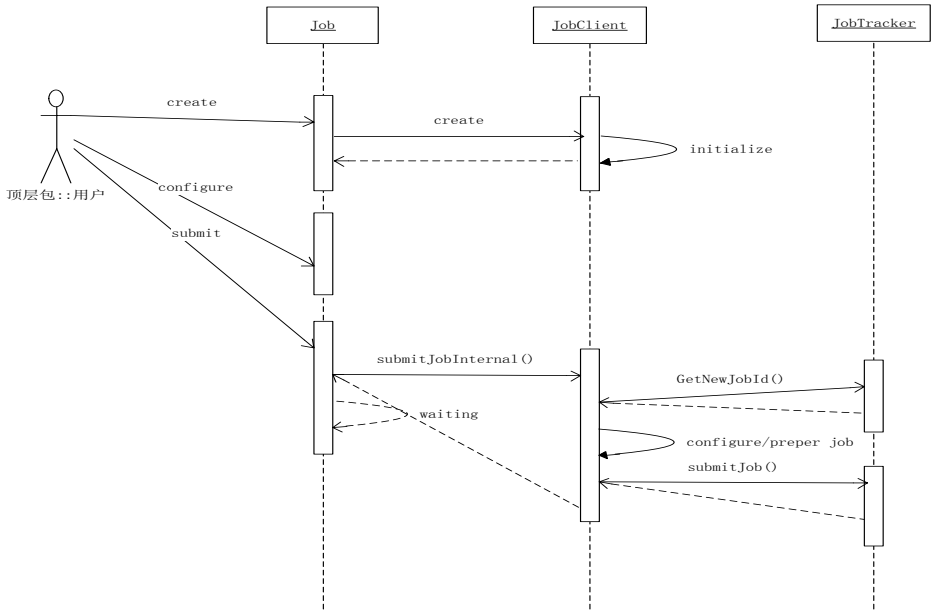

图 2-4 MapReduce 程序提交 Job 时序图

2.2 Hive 介绍

2.2.1 Hive 简介

Hive[23,24]是一个数据仓库系统，它是基于 Hadoop 框架基础上开发的。它最早是 FaceBook 公司为了处理海量的日志和用户数据而提出来的。Hive 的主要目标就是使得熟悉 SQL 但是不熟悉编程的人员能够很好的使用去分析数据。Hive 提供了许多操作工具，用来进行数据 ETL 操作。Hive 有自己的规范语言类似于数据库中的 SQL 查询语言，称为 HiveQL，让熟悉数据库 SQL 语言的用户通过 HiveQL 语言查询数据。同时，Hive 也允许熟悉 MapReduce 的人员开发自定义的 MapReduce 来处理更复杂的分析工作。

日志数据、关系型数据库的数据，这都是结构化的数据，这些数据可以存放在类似于表的结构里。Hive 对于大数据同样提供了存储这些结构化数据的方式。Hive 从关系型数据库中引入了许多的类似的概念，如表、行、列、模式等。此外，Hive 还引入了元存储（metastore）的概念用来表示 Hive 数据库的基本信息，元存储一般存储在一个关系型数据库中，如 Mysql 或 Oracle 等。

2.2.1 HiveQL 介绍

下面介绍一些常用的 HiveQL[25]操作。

（1）创建表

在 HiveQL 中创建表的完整语法为：

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] tablename

[(colname datatype [COMMENT colcomment], ...)]

[COMMENT tablecomment]

[PARTITIONED BY (colname datatype [COMMENT colcomment], ...)]

[CLUSTERED BY (colname, colname, ...)

[SORTED BY (colname [ASC|DESC], ...)] INTO numbuckets BUCKETS]

[ROW FORMAT row_format]

[STORED AS file_format]

[LOCATION hdfs_path]

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] tablename 创建一个表并指定表的基本信息包括表名，字段名，字段类型，以及字段的注释。

PARTITIONED BY 指定划分列。

CLUSTERED BY 指定桶的信息。

SORTED BY 指定排序字段。

ROW FORMAT 指定数据存储格式。

STORED AS 指定保存的数据文件格式。

LOCATION 如果创建的是外部表，指定文件路径。

（2）导入数据

当表创建好了之后，需要把数据导入到表中，Hive 不支持 insert 操作，用户可以使用 LOAD DATA 命令把文件中的数据导入到 Hive 表中。

LOAD DATA [LOCAL] INPATH filenamepath [OVERWRITE] INTO TABLE tablename [PARTITION (partcolname1=value1, partcolname2=value2 ...)]

上述命令是将 filenamepath 文件中的内容导入到 tablename 表中。如果带有 OVERWRITE 关键字表示新的内容会覆盖原来表中的数据，不带有表示追加到原数据的后面。带有 LOCAL 关键字表示数据文件在本地文件系统，不带有则文件在 HDFS 文件系统中。

（3）查询数据

大多数情况下，HiveQL 查询方式都和数据库 SQL 语言类似。但是，HiveQL 查询结果集有可能会非常庞大，需要把结果集保存到其他地方，如 Hive 表，HDFS 中或本地文件系统中。

INSERT OVERWRITE destination

SELECT col_list FROM from tablename

[WHERE condition]

[GROUP BY col_list]

[ORDER BY col_list]

（4）操作符、函数

和 SQL 语言一样，HiveQL 语言也支持多种操作符和函数，支持关系型运算符、算数运算符、逻辑运算符等。函数包括数学函数、聚合函数、字符串函数、日期函数等。同时 HiveQL 也支持自定义函数，用户可以根据需求自己定义所需函数。

2.3 Flume 介绍

Apache 的 Flume[26]架构是一个可靠的、分布式的、具有高可用性的日志收集工具，它能够快速有效的收集、聚合各个服务器的日志，并把收集的这些大量的日志数据传输到指定的目的数据仓库中。Flume 支持多种不同的数据源，如控制台、文本文件、系统日志文件等等。同时也支持多种目标数据存储系统，如文本文件、HDFS 分布式文件等。

Flume 数据传输方式是以数据流作为传输的载体，其本质就是作为一个数据传输通道，把数据从初始端传输到目的端。图 2-5 为 Flume 数据流模型。

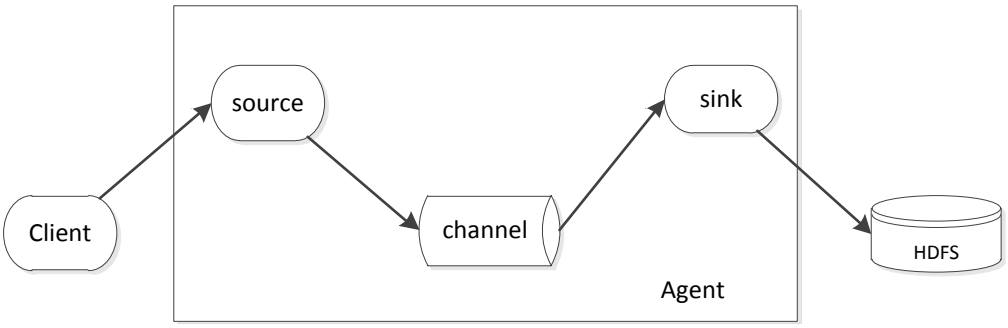

图 2-5 Flume 数据流图

Agent 是 Flume 的主要组成部分，是一个独立的进程，它能够接收数据并把数据传输到指定的位置。一个 Agent 包含 source、channel、sink 三个重要部分。Source 是用来接收数据的，是数据传输的开端。Channel 顾名思义是传输数据的通道，接收来自 source 端的数据并且进行短暂的存储，等待 sink 进程的获取，channel 可以为内存方式、JDBC 方式等多种。Sink 拿到 channel 的数据之后会把数据传输到指定的目的地文件系统或者下一个 source 继续进行数据的传输，然后会移除 channel 中相应的数据。Channel 支持 source 和 sink 这种数据交换模式类似于经典的生产者—消费者模式，source 作为生产者生产数据，channel 作为 source 生产的产品存储的仓库，而 sink 作为消费者，只要 channel 有产品，sink 就会消费，这能够合理、有效的利用系统资源。

Flume 支持一个 source 传输给多个 channel，数据最终到达多个目的地的数据传输模式。图 2-6 展示的是 source 把数据写入多个 channel 的过程。

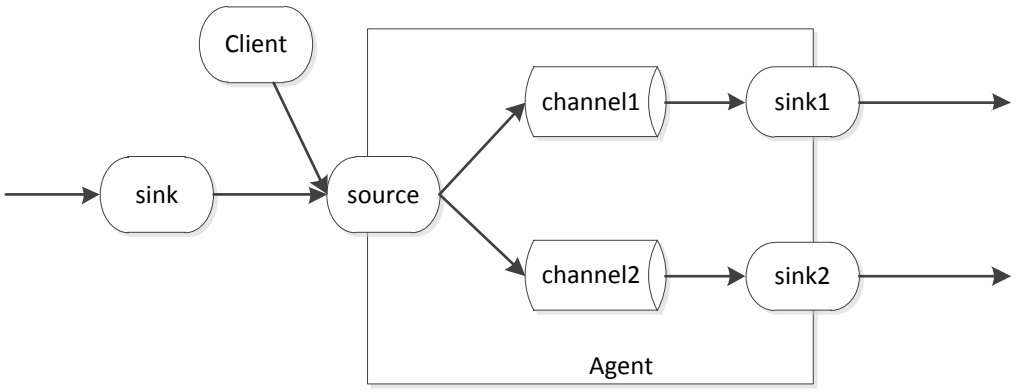

图 2-6 数据写入多个目的地流程图

还有另外一种情况，Flume 实现数据的聚合能够把多个 source 汇总到一个 channel，图 2-7 是一个数据聚合的数据流图。

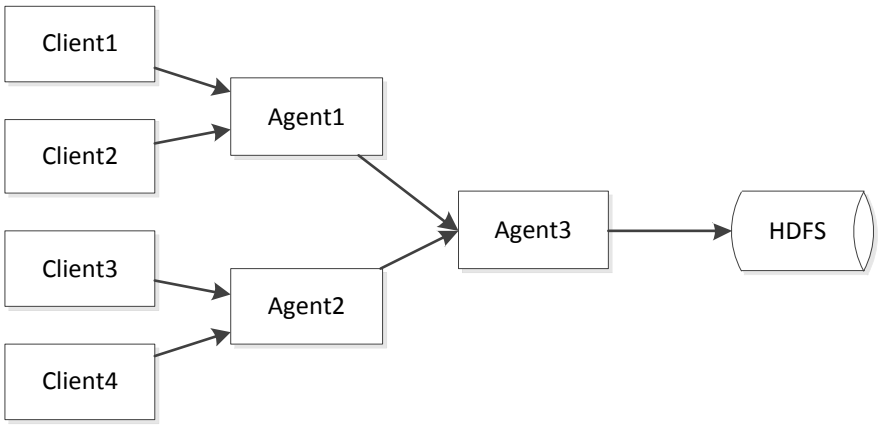

图 2-7 数据聚合流程

Flume 这种通过 Agent 进行数据传输的方式保证了信息的传递性。数据从一个 Agent 传递给下一个 Agent 的时候，Flume 会启动两个进程，前一个 Agent 的 sink 负责把数据写入下一个 Agent 的 source 端，后一个 Agent 接收到数据之后进行下一步的传输。后一个 Agent 的在事务提交之后会给前一个 Agent 返回一个成功接收的信息，前一个 Agent 接收到成功信息后才会提交自己的事务。这种方式确保了数据在传输过程中不会出现丢失。这种数据传输机制也是 Flume 失效后数据得以恢复的基础。当数据在多个 Agent 之间传输过程中出现错误时，前一个 Agent 会缓存这些数据，后面 Agent 恢复之后，数据能够继续传输。但是这种缓存是被动的，如果长时间的没有恢复，最终很有可能会导致数据的丢失。图 2-8 是两个 agent 之间传递消息成功的时序图。

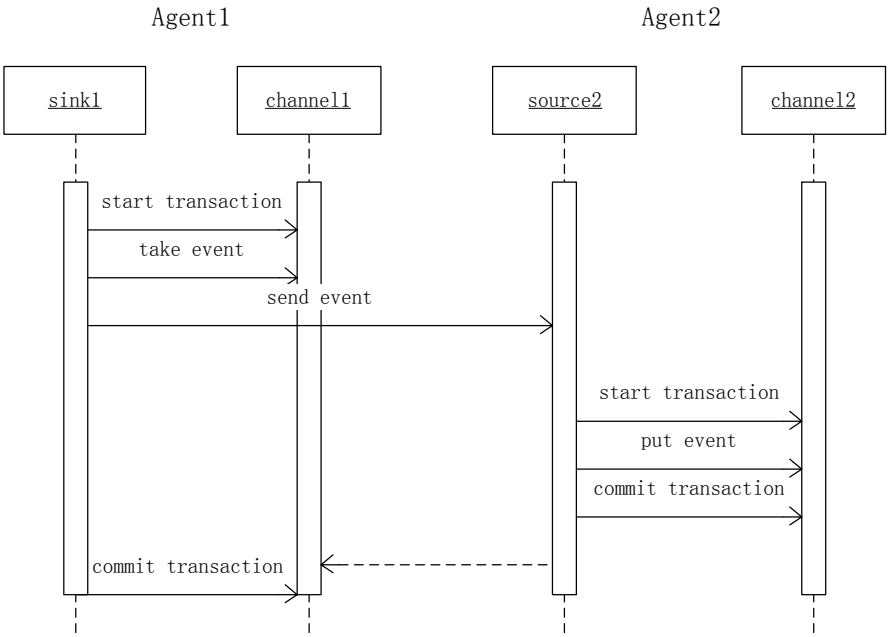

图 2-8 两个 agent 交换信息的时序图

2.4 Redis 介绍

Redis[27,28]是 NoSql 数据库家族的的一名成员，是一个高性能的 Key-value 存储的内存数据库。它把数据存储在内存中，在应用系统和数据库之间提供高性能的数据缓存服务。Redis 系统支持很多的数据类型，主要有字符串类型、集合、列表、哈希列表等等，同时也支持这些数据类型各种操作，例如，push、pop、add、remove 以及交集、差集等操作，这些操作都是封装好的，我们只需调用接口函数即可使用。

Redis 是内存数据库，但是内存会由于断电等原因数据会丢失。所以 Redis 支持数据的持久化，数据持久化简单的说将就是断电后数据可以进行恢复，不会丢失数据，也就是说需要把 Redis 内存中的数据备份不会丢失的地方，如磁盘，来确保数据的持久性。Redis 数据持久化支持两种方式，一种是 RDB 快照的方式，这种方法是把 Redis 的数据的快照存储成一个二进制文件。Redis 借助 fork 命令中的 copy on write 机制，在生成快照的时候，将当前进程分出一个子进程，然后在子进程中循环的读取所有的数据，最后将数据写成 RDB 文件。持久化的频度可以人为的设定，例如，可以设置 Redis 在 m 秒内如果有超过 n 个 key 的数据发生了改变就自动进行备份。但是，Redis 出现各种异常时，从 RDB 文件恢复的数据并不一定是最新的，从上次执行快照到现在的时间内，Redis 中修改的数据就会丢失，这是快照持久化最大的缺点，对于一些数据要求严格的业务这种方式就无法容忍了。

Redis 还支持另一种持久化方式，Append-only file[23]日志的方式来实现持久化，简称 Aof。Aof 比 RDB 快照方式持久化数据更全面，在 Aof 方式下，Redis 通过 write 函数会把每一次的数据操作都记录在日志文件中。Aof 文件是可读取的文本文件，文件的内容就是 Redis 标准命令。当 Redis 因为失效重启时直接重新执行追加文件记录中的操作就可以恢复 Redis 中最新的数据。这种方式很好的保证了数据的一致性，但是也带来了一定的影响。首先，每一次操作都会写入到 Aof 文件，性能会有影响。其次，随着不断的追加，Aof 文件会越来越大，数据恢复时执行过程漫长。基于这种情况，Redis 也提供了很好的解决方案，由于 Aof 文件中同一个 Key 可能会有很多的操作，我们只需要 Key 的当前状态就可以，所以在后台可以对 Aof 进行重建，把当前的 Key 的命令写入到 Aof 文件即可，这样大大缩小了 Aof 文件的大小，同时也保证了数据的完整性。

2.5 本章小结

本章主要介绍了系统中所涉及的架构的理论知识，着重介绍了 Hadoop 架构的应用。介绍了 Hadoop 的基本体系，HDFS 分布式文件系统和 MapReduce 模型的基本概念。Hive 工具的优势以及语法知识。Flume 架构的应用场景以及作为日志传输工具的优势。Redis 作为内存数据的优势等。通过以上的技术的研究，为以后的系统实现提供了理论基础。

第三章 需求分析与架构设计

本章介绍了系统的需求以及架构的设计。从系统的功能性需求和非功能性需求方面出发，介绍了整个系统的所要实现的功能，以及性能指标。利用现有技术，设计了系统的整体架构和各个功能模块，并对各个功能模块有个详细介绍。

3.1 需求分析

在传统的数据分析平台中，数据一般存储在传统的关系型数据库中如：Oracle、Mysql、SQL Server 等数据库中。关系型数据库具有较强的语义表达能力和较强的数据之间的关联能力，因此，能较好的满足分析人员的需求。但是，随着“云计算”时代的到来，信息社会的不断发展，数据量呈爆炸式的增长。互联网应用服务也越来越庞大，服务器达到了成百上千台。公司也意识到各个服务器产生的有价值数据越来越多，运营人员分析需求也在增加。以淘宝网为例，淘宝网每天都有几十亿的店铺、商品浏览记录，上千万的成交记录。当数量成为“海量”的时候，数据的存储、计算、查询能力在传统的关系型数据库中就显得力不从心了。

3.1.1 功能需求

系统主要功能是为数据分析人员提供数据统计支持。主要提供用户管理、查询、报表展示、数据监控等功能。

1. 实现数据收集、汇总
   a) 自动收集各个应用服务器和系统服务器产生的日志。
   b) 自动抽取数据服务器的数据库中数据。
   c) 自动把数据统一放到 HDFS 中。
2. 数据统计。
   a) 用户信息

用户是一个产品、一个企业赖以生存的基础。只有更好的了解用户才能更好的抓住用户，为用户服务。用户信息主要包括，用户基本信息，用户安装的产品信息以及用户的行为信息。

b) 产品信息

产品是企业发展的基础，只有有了适合用户的产品才能吸引更多的用户。产品信息包括，产品类别，版本信息，渠道信息等。

c) 收入信息

收入是衡量一个企业收支情况的一个标签，是企业的根本。收入越多说明用

户对产品越肯定。收入信息包括，收入总额，分国家、产品收入，支出结算。

d) 用户行为信息

用户行为信息是分析用户喜好以及收入的关键。用户行为包括，安装，联网，更新，卸载以及付费等。根据用户的行为我们可以分析用户的喜好以及用户的分布情况等。

e) 推送信息

推送信息是根据用户的喜好制定的，不同的用户推送不同的产品，从而提高用户数量以及用户质量。用户推送是根据用户行为信息得到用户的特有喜好，从而推给用户产品。

3. 数据展示

a) 报表展示

根据需要定期查询的数据项作为报表展示出来，这样就可以减少每次都要查询的工作量。主要包括日报、周报和月报。报表查询人员选择所需数据项进行数据查询。

b) 数据查询功能

一些特殊需求不是经常查询的或者对于熟悉 SQL 的人员提供数据查询接口，可以实时查询。

c) 数据结果下载

系统使用人员可以对查询的数据结果进行下载，以及历史查询结果的下载。

4. 权限管理

a) 用户报表权限管理

主要管理用户可以查询哪些数据库表，用户可以查询哪些报表。

b) 用户维度权限管理

主要管理用户可以查询哪些具体维度值的数据。

c) 用户信息管理

增删用户。

5. 监控功能

a) 数据流监控

数据在各个服务器中传输过程进行监控，防止数据在传输中丢失。

b) 数据异常监控

数据分析中数据量的横向和纵向对比，超出阈值进行报警。

c) 服务器监控

监控服务器的运行状况。

d) 报警功能

数据或服务器出现异常进行报警（邮件、短信方式）。

3.1.2 非功能性需求

1. 响应时间
   a) 非实时数据，今天处理昨天的数据。
   b) 实时数据，一分钟之内处理完成。
   c) 用户查询，一般情况一分钟之内响应。数据量很大的话可以延长响应时间。
   d) 用户并发查询，响应时间可相对延长，但也要控制在 5 分钟之内。
2. 系统可靠性
   a) 可以允许单点故障。
   b) 数据出现异常能够及时报警，并重新计算。
3. 可长时间保存历史数据。
4. 良好的可扩展性，需求增加时可以迅速扩充。

3.1.3 用例分析

一、用户角色

该系统根据用户权限分为以下四种角色：

1. 普通用户

普通用户一般为普通员工，拥有查询权限，查看以及修改个人信息权限，同时可以查看所分配的报表权限。

2. 高级用户

高级用户一般指部门经理级别，可以查看部门员工信息，查看所有报表，以及报表权限的分配。

3. 系统管理员

系统管理员负责系统的设置，系统权限管理，用户管理，以及系统硬件设备管理。

4. 系统开发人员

主要负责系统的开发维护，新增数据处理，数据故障处理，新增报表以及下线不再需要的数据和报表。

二、用户用例

1. 系统整体用例

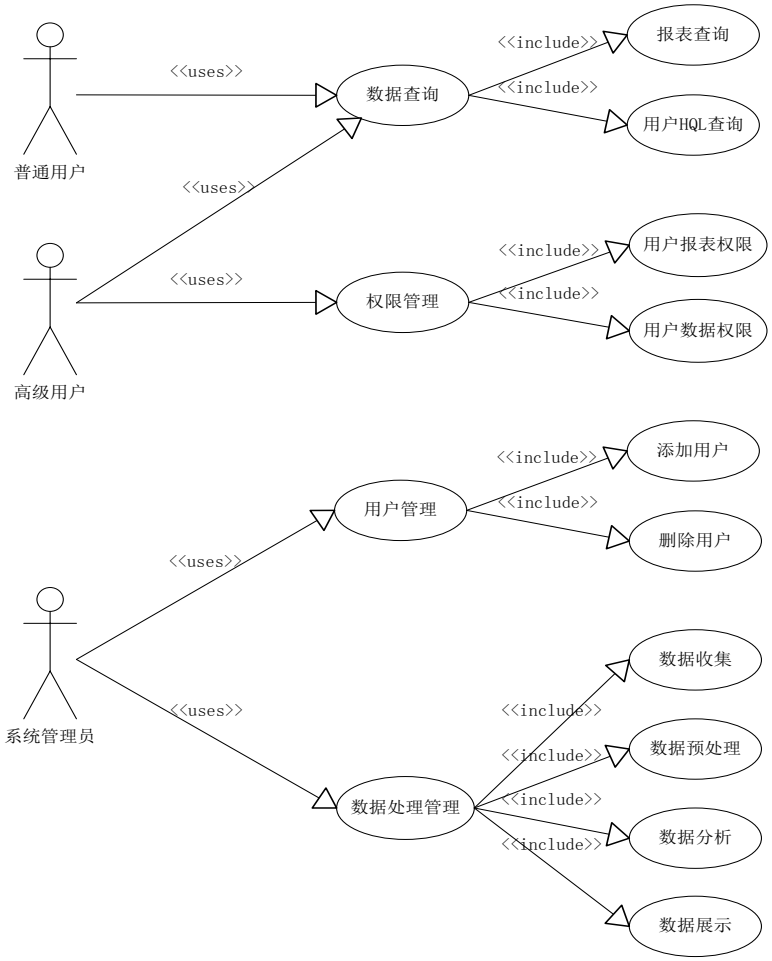

图 3-1 系统用例图

2. 用户管理用例

用户可以查看自己的个人信息，同时修改登录密码。

基本事件流如下：

1）显示用户基本信息。

2）选择修改密码。

3）输入初始密码。

4）输入新密码 2 次。

5）确认。

前置条件：用户已登录且初始密码正确。

后置条件：新密码为有效字符。

分配权限基本事件流如下：

1）选择需要分配权限的用户。

2）选择需要的报表。

3）选择新增（删除）。

4）确认。

前置条件：用户已登录且为高级用户。

后置条件：无。

系统管理员主要负责新增用户，维护系统正常运行。

新增用户基本事件流如下：

1）选择新增用户。

2）输入用户名及初始密码。

3）选择初始报表权限。

4）确认。

前置条件：用户已登录且为系统管理员。

后置条件：无。

3. 报表管理用例

报表主要是运营人员由运营人员使用，负责查看报表信息，总结产品使用情况以及收入结算。针对一般员工报表只有查询功能。只有报表管理人员才可以根据运营需求对报表进行修改，增加，下线操作。

查询报表基本事件流如下：

1）显示可以查看的报表

2）根据查询条件显示报表信息

3）可根据需要下载报表到 csv 文件。

前置条件：用户已登录。

后置条件：无。

3.2 架构设计

本系统就基于应用服务器日志和数据服务器的数据库构建的数据处理系统，其基本目标就是实现通过对数据的收集，预处理，分析，进而展示的过程，生成各种产品使用情况的报表和统计数据，为运营管理提供有效支持。

本系统是基于 Hadoop 的分布式架构，可以解决海量数据存储的问题，提供自动容错机制，利用分布式计算的特性，及时响应计算需求。

数据分别从日志服务器，数据服务器，以及 PC 机等抽取数据，传输到 Hadoop 的 HDFS 文件系统，经过 Hadoop 以及 Hive 进行分析处理，最后传输到关系型数据库或存储到 Hive 表中，用户可以通过 Web 界面进行报表查询或者直接使用 HiveQL 语句进行查询。

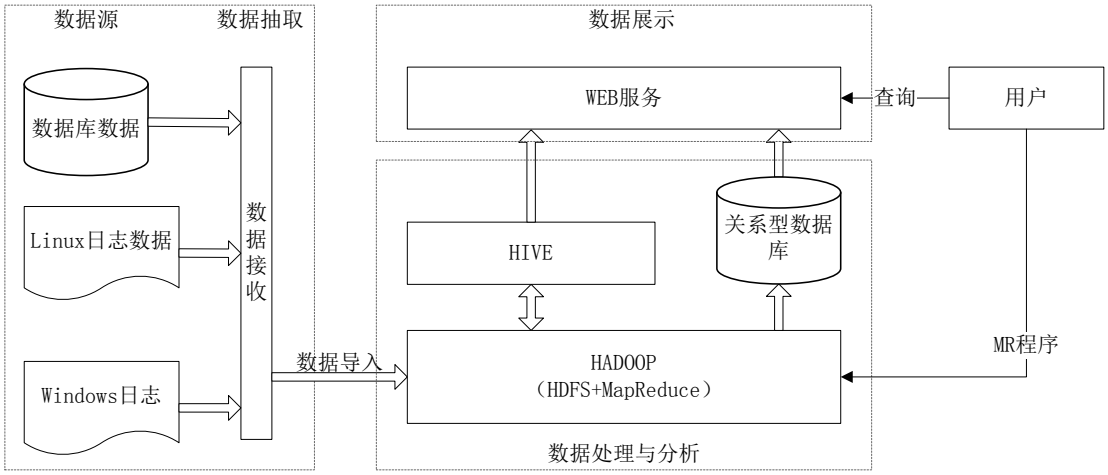

图 3-2 系统总体架构图

通过上述架构图可以看出，本系统主要包括数据收集模块，数据预处理模块，数据分析模块以及数据展示模块。

3.2.1 数据收集模块

数据收集[29]是指从各个应用服务器，数据服务器中把数据收集到一起，然后进行处理。对于一个企业来说，拥有着几十上百台的应用服务器，记录的数据的种类繁多，格式不一，每天产生的数据量很大。如果把这些数据都集中到一台服务器上进行分析，会对服务器产生很大的负担，所以我们采用分布式的文件系统来代替传统的单机模式来保存产生的数据。

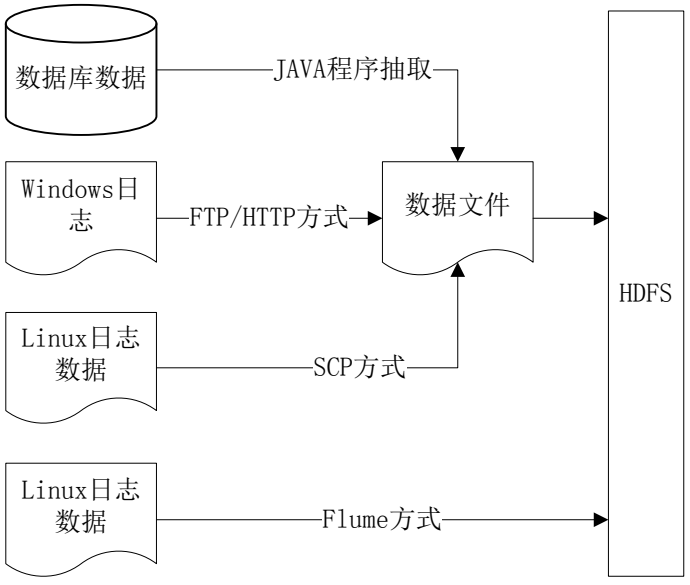

图 3-3 数据收集架构图

本系统采用 3 中方式来处理不同情况下的数据收集。

- 针对于 windows 系统服务器，这种数据量比较小，我们采用 ftp 或 http 服务的方式进行数据传输。
- 对于业务的日志数据分为实时和非实时两种情况。非实时数据采用每天

数据放入到 NFS 中，然后 SCP 到数据收集服务器。实时数据采用 Flume 的方式进行实时写入到 HDFS 中。

◆ 对于数据服务器中数据库中的数据，采用每天定时抽取到文件中然后 SCP 到数据收集服务器。

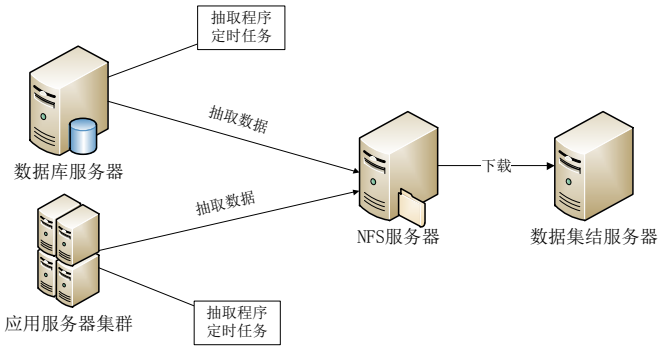

图 3-4 非实时数据收集流程图

日志中的同一服务器可能会记录多种样式的日志，同一种日志数据记录在不同的服务器上，需要把同种样式的日志传送到 HDFS 中的同一文件中。对于 SCP 方式我们采用 NFS 文件系统。日志的命名有相应的规范手册，相同类型的日志命名为相同日志标识开头的文件。对于 Flume 方式，每一台服务器上都会记录多种日志，对于同一个 APP1，会为其配置多个接受数据的 agent 来接受不同类型的日志文件。对 APP1 和 APP2 中产生同种日志，每个服务器上有单独的 Agent 接受以后会聚合到一个新的 Agent，最后把数据统一进行传输到 HDFS 文件系统中。

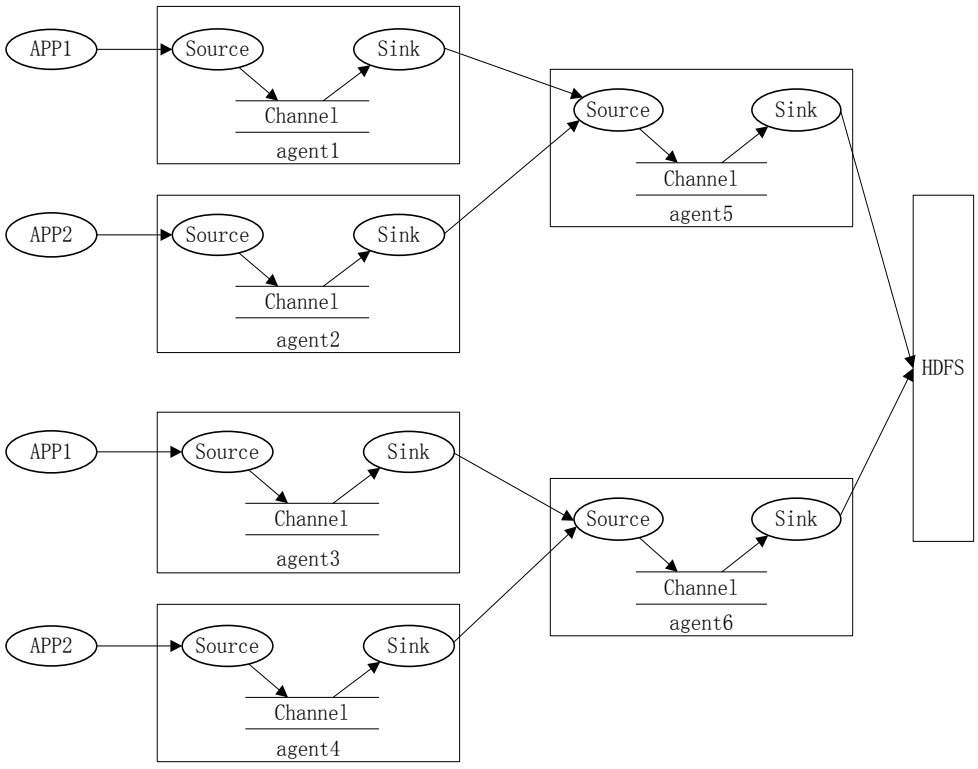

图 3-5 Flume 收集流程图

3.2.2 数据预处理模块

数据预处理[30]模块是关系到数据结果质量以及数据分析效率的关键步骤，作为数据处理的开端，数据预处理模块是本系统的重要部分。在本系统中，数据预处理模块主要包括三个部分：数据清洗，数据集成，用户数据初始化。这几个部分先后作用于原始数据，最终为数据分析步骤准备好待处理的数据。

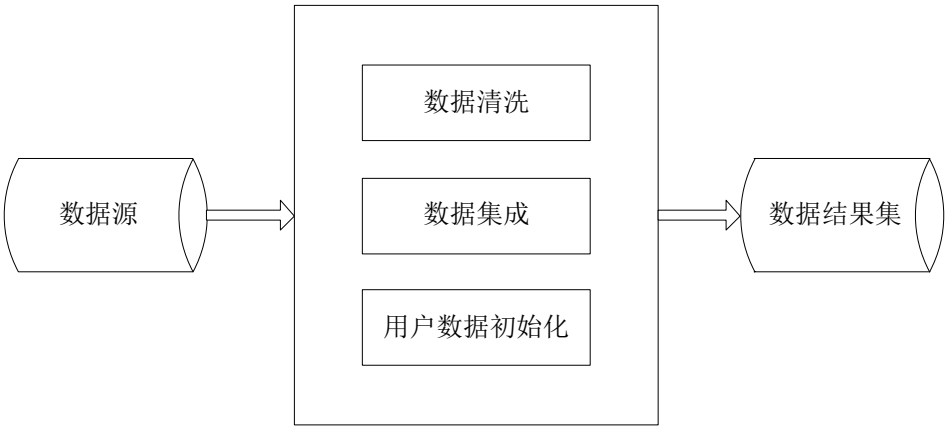

图 3-6 数据预处理模块架构图

数据清洗[31]主要指的是对原始数据中多种产品不同的配置格式进行统一，包括定义每个字段的含义以及统一分隔符，同时还会去掉一些记录不完全的坏数据。保证数据的格式统一，信息完整。

数据集成是把来自各种不同数据源的数据集成到一起，保证数据格式的统一。

用户数据初始化是通过用户标识把用户的基本属性同用户的行为关联起来，成为一条信息交完整的数据。

并非所有的数据都需要经过这些清洗步骤，如数据服务器中数据库中的数据本身就是规范化的数据，无需进行数据清洗就可以直接分析使用。可以根据具体的数据需求，选取其中的环节进行数据的变换，达到加快数据分析的目的。数据预处理环节是提高数据分析效率的保证。

3.2.3 数据分析模块

数据分析模块主要功能是完成数据的分析处理。在本系统中数据分析主要包括用户行为分析，各产品用户数分析，产品、用户、国家等关联分析。在此，本系统主要利用 MapReduce 任务完成数据分析的工作。通常一个复杂的数据分析任务，会包括很多的子任务，此时，就会有一个主进程和多个子进程结合在一起共同完成整个复杂分析的任务。同时系统也采用 HiveQL 语言分析数据，HiveQL 语言自动转化为 MapReduce 程序处理，整个 MapReduce 程序是封装好的。

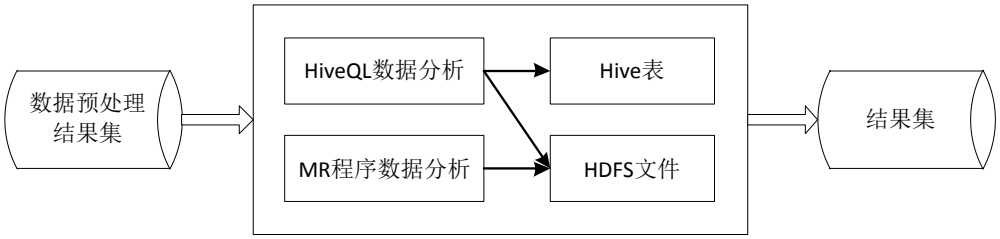

图 3-7 数据分析模块架构图

数据分组统计和数据关联查询是最常用的数据分析方式，是了解用户行为的重要手段。

3.2.4 监控告警模块

监控告警模块负责数据流的监控、数据处理流程监控、数据粒度监控、服务器监控。监控告警模块是整个系统的保障，通过对各个环节进行监控保证了系统的稳定性以及数据的准确性。

数据流监控，主要负责监控数据在各个服务器传输过程中的一致性，数据完整性。

数据处理流程监控，主要负责数据预处理和数据分析过程中，处理任务是否出现异常情况。

数据粒度监控，主要监控数据一段时间内的变化情况是否在阈值范围内。

服务器监控，主要监控服务器的运行状况。

3.2.5 数据展示模块

数据展示主要分为报表展示和 SQL 查询，针对大结果集的数据我们保存到 Hive 表中，分析人员可以直接查询表中的数据或者表中数据以报表形式展示出来。对于数据结果集不大的数据我们直接从 HDFS 通过 sqoop[32]导入到关系型数据库中，数据最终以报表的形式展示出来。

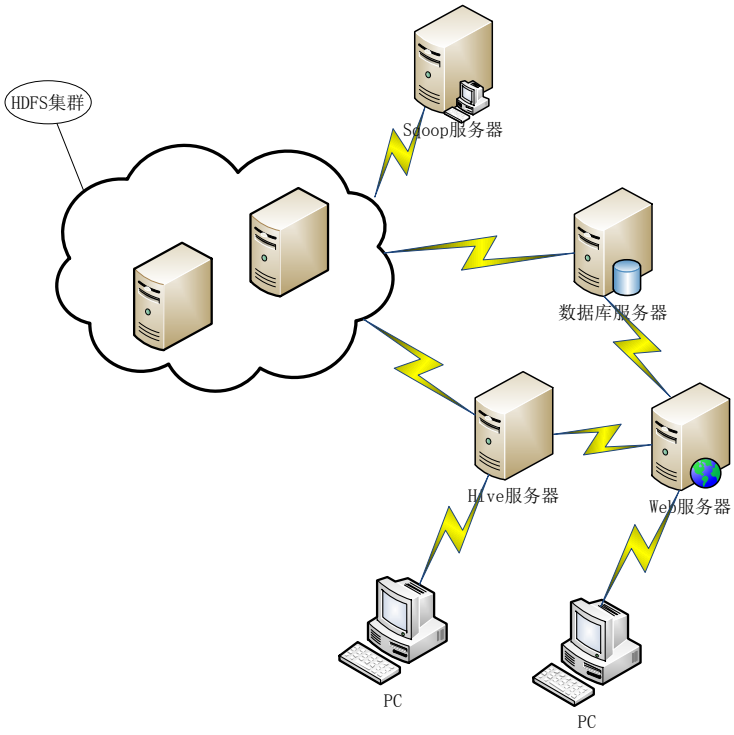

图 3-8 数据展示模块架构图

数据主要是通过 Web 界面方式展现给用户，用户也可以通过查询 Hive 表的方式查询数据结果。

3.3 本章小结

本章在大量的技术研究基础之上，对整个系统进行了详细的需求分析，提出了整个系统实现的具体目标。针对系统需求设计出了系统的整体架构以及各个功能模块，并且对各个功能模块做出了详细的描述。

第四章 详细设计与实现

本章在前面章节的基础上，搭建了系统的硬件设施，本系统采用 Flume 数据的实时传输和非实时数据的隔天传输两种方式进行数据收集，并把数据集中放入 HDFS 文件系统中。通过 MapReduce 程序和 HiveQL 语言分析方式进行数据分析计算，并把小结果集数据通过 Sqoop 导入到关系型数据库中，大结果集数据保存到 Hive 表中，数据以报表的形式展示出来。给运营人员运营决策提供数据支持。

4.1 数据收集模块

数据收集模块是整个系统的基础，如何将各个服务器上大量的数据传输到 HDFS 文件系统是本系统的一个瓶颈，如果传输时间过长则影响整个系统的效率。本系统采用实时传输的 Flume 方式和非实时传输的 SCP 传输方式进行数据的传输。

4.1.1 数据的生成

本系统中数据分为两部分数据，日志数据和数据库数据。日志数据记录在日志服务器上，每天生成一个日志文件。数据库数据直接写入到 Oracle 数据库中，每天进行数据同步。

日志数据主要采用 Apache 的 Log4j[33]开源插件来实现应用服务器日志记录功能。Log4j 可以分级别的记录不同类型的日志，优先级从低到高依次为 DEBUG、INFO、WARN、ERROR，通过在这里定义的级别，根据需要定义应用服务器中记录日志信息的级别。例如我们定义了 INFO 级别的日志，只有等于或者高于这个级别的才会进行处理，则应用程序中所有 DEBUG 级别的日志信息将不被打印出来。同时 Log4j 支持 Flume 实时写入到 HDFS 文件系统。本系统对于非实时日志，采用 Log4j 写入本地文件系统，然后定时拷贝的方式。对于实时日志，一方面采用 Flume 实时把日志写入到 HDFS 中，另一方面在本地文件系统也记录一份，防止 Flume 传输失败，以便能够及时恢复数据。

Log4j 写入本地文件系统配置包括，日志输出方式，DailyRollingFileAppender 为每天产生一个日志文件；日志保存路径；日志输出信息格式以及日志输出级别。

应用服务器日志有些数据量比较大，每天一个文件处理起来效率比较低，这种日志采用按小时来分割文件，每个小时一个文件进行处理。

日志记录有一定的规范，以便在后续的数据处理中方便、高效。根据不同业务创建不同目录，相同业务的日志放到同一目录中。同一业务的不同日志命名以不同日志关键字开头，例如：

101_{业务}_{日志描述}_DailyRolling.log.${TODAYNAME}。

日志内容格式举例：

YYYYMMDDHHMMSS - INFO/ERROR - CommandID - BUSINESSID – Content

Log4j 配置

Log4j 应用于 Flume 实时传输的日志的配置

log4j.appender.flume= org.apache.flume.clients.log4jappender.Log4jAppender

log4j.appender.flume.Hostname = 192.168.35.81

log4j.appender.flume.Port = 41414

log4j.appender.flume.layout = org.apache.log4j.PatternLayout

log4j.appender.flume.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} - %p - %m%n

log4j.logger.myclass =info,flume

Log4j 应用于非实时传输的日志配置

log4j.appender.dailyrollingfile=org.apache.log4j.DailyRollingFileAppender

log4j.appender.dailyrollingfile.file=D:/BOSS_DailyRolling.log

log4j.appender.dailyrollingfile.DatePattern='.'yyyy-MM-dd

log4j.appender.dailyrollingfile.layout=org.apache.log4j.PatternLayout

log4j.appender.dailyrollingfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %c{1}:%L %5p - %m%n

log4j.logger.myclass=info,dailyrollingfile

Flume 使用中需要使用 Apache 的 org.apache.flume 下面的类，我们需要导入一些特定的 jar 包，（例如：flume-ng-sdk-1.2.0.jar，flume-ng-log4jappender-1.2.0.jar 等）。从上面的配置来看，Flume 数据传输主要配置两个属性，Hostname 是运行 Flume Agent 的服务器的地址，Log4j 写入的 source 只能是 avro source，Port 就是启动的 avro source 所监听的端口。

数据库数据记录，采用前端APP应用直接连接数据库，直接对数据库进行操作写入数据。

4.1.2 数据传输

数据传送分为日志数据传输和数据库数据传输。日志数据传输分为 Flume 方式传输和拷贝方式传输。

Flume 方式：

Flume 方式传输是实时的把数据信息写入到 HDFS 中以便进行实时数据处理。这里我们把 Flume 作为一个数据通道。我们使用的 Flume1.x 版本。Flume 需要

创建自己的 Agent。Agent 主要包括 source，channel 和 sink 三个组件，支持 Log4j 写入的 source 只能是 avro source，我们采用 HDFS 分布式文件系统作为数据存储系统，所以使用 HDFS sink，为了保证数据的安全性，我们使用 File channel 作为数据传输通道。

下面是一个 Agent 配置的例子：

Agent 的配置

```
agent-1.sources = avro-source
agent-1.sinks = hdfs-sink
agent-1.channels = file-channel
# set channel for sources,sinks
agent-1.sources. avro-source.channels = file-channel
agent-1.sinks. hdfs-sink.channel = file-channel
# properties of avro-source
agent-1.sources.avro-source.type = avro
agent-1.sources.avro-source.bind = localhost
agent-1.sources.avro-source.port = 10011
# properties of file-channel
agent-1.channels.file-channel.type = file
agent-1.channels.file-channel.checkpointDir = /home/flume/channel/checkpoint
agent-1.channels.file-channel.dataDirs = /home/flume/channel/data
agent-1.sinks.hdfs-sink.type = hdfs
agent-1.sinks.hdfs-sink.hdfs.path = hdfs://namenode/flume/business/
agent-1.sinks.hdfs-sink.hdfs.filePrefix = flume_
agent-1.sinks.hdfs-sink.hdfs.round = true
agent-1.sinks.hdfs-sink.hdfs.roundValue = 24
agent-1.sinks.hdfs-sink.hdfs.roundUnit = hour
```

这里配置了一个 Agent 叫 agent-1，包括一个 avro source，需要设置它的几个属性，type、bind 和 port，type 表示的是 source 的类型即 avro source，bind 表示它监听的 IP 或主机名，port 表示监听端口，bind 和 port 加起来就是产生日志的应用服务器。Channel 采用的是 File channel 的方式，File channel 可以保证数据在传输过程中的一致性。File channel 必须定义一个 type 属性，当多个 Agent 使用 File channel 作为传输通道的时候就需要手动 checkpointDir 和 dataDirs，checkpointDir 保存检查点的路径，dataDirs 保存 channel 数据文件的路径，默认会有一个路径，但是如果多个 Agent 的时候会有一个 Agent 初始化，然后使用该

路径，其他 Agent 就会初始化失败。Sink 使用的是 HDFS sink，把数据最终保存到 HDFS 文件系统中，需要配置 type 属性值为 hdfs，path 属性为 HDFS 文件的保存路径，round 属性表示文件滚动属性，上面配置的是 24 小时自动滚动的方式，即 24 小时产生一个新文件。

需要经过多个 Agent 进行传输的数据，前一个 Agent 的 sink 作为后一个 Agent source 端的写入方，多个 Agent 之间都是通过 avro sink 和 avro source 进行数据的传输。

拷贝方式：

应用服务器每天会产生大量的日志，我们需要把数据分析所需的日志提取出来，放到 NFS 中进行数据拷贝。相同类型的日志我们都以相同的日志关键字开头进行命名。例如：101_PI_useractivate_DailyRolling.log.${TODAYNAME}表示用户激活日志，101_PI_userupdate_DailyRolling.log.${TODAYNAME}表示用户更新日志。都表示用户行为信息日志，所以都以 101_为开头命名。我们放入 NFS 中没有必要两个日志进行传输，可以进行合并。然后把每天的文件进行 tar 打包，最后以 SCP 的方式传输到 Hadoop 服务器。有些日志数据比较特殊，不是在 linux 服务器上，而是在 Windows 服务器上，这种我们采用 ftp 下载或者 http 方式进行数据传输。

数据库数据采用 Java 程序抽取方式进行数据抽取。每天定时执行 Java 进程进行数据抽取，最后写入到 HDFS 文件系统中。Java 程序抽取方式包括全量抽取和增量抽取。

全量抽取

读取整个表的数据，然后写进文件中。全量抽取这种情况主要针对一些配置表和数据量较小的表，同时还有一些表中的数据无法区分是否是增量的情况。全量抽取数据可以保证数据完整一致。

增量抽取

识别新数据，写进文件中。增量数据的表中都有时间字段作为判断依据，一般都是凌晨获取昨天的数据。增量数据抽取都是抽取前一天的数据，这种数据分析有一定的延时性，但是对于一些情况是可以容忍的。

Java 程序会通过读取数据库中一张配置表来判断该表数据抽取方式。

表 4-1 抽取配置表结构

TABLE_SYNCTABLE 抽取数据配置表		
字段名	字段类型	说明
SYNC_TABLENAME	VARCHAR2(30)	表名
SYNC_COLUMNNAME	VARCHAR2(200)	增量抽取依据字段名，若留空，则表示全量抽取
SYNC_QUERY	VARCHAR2(30)	抽取字段集
SYNC_ADDDATE	DATE	添加时间

字段详细解释：

SYNC_TABLENAME

表名。需要抽取的数据库表的名称，不包括表的用户名。每个数据库下面都会有一张这样的配置表，数据抽取程序只需要配置抽取的数据库的名称和用户名称即可。

SYNC_COLUMNNAME

增量抽取依据的字段名。通过判断该字段来识别表的抽取模式，字段值为空则表示该表需要全量抽取，不为空则表示增量抽取。例如该字段的值为LAST_DATE，则表示抽取时会按照该表的这个字段来区分增量数据。在抽取时查询增量数据的 where 条件即：where LAST_DATE>=TRUNC(SYSDATE)-1 AND LAST_DATE<TRUNC(SYSDATE)。此时抽出的数据就是前一天新增的数据。每个表的区分字段都会不同，但通常情况下都是以时间字段作为区分的，特殊情况会以某个字段的字符串的值作为区分。

SYNC_QUERY

抽取字段的集合。有些表的我们并不是需要抽取所有的字段用来数据分析，仅仅定义几个需要进行分析的字段即可。SYNC_QUERY 就是用来配置抽取的表的特殊字段的。如果 SYNC_QUERY 为空则表示抽取全部字段，否则则按照配置的字段进行抽取。例如一个表中有字段四个字段 COLUMN_A、COLUMN_B、COLUMN_C、COLUMN_D。在抽取的时候我们只需要抽取 COLUMN_A、COLUMN_C、COLUMN_D，那么 SYNC_QUERY 在配置该表时值就是“COLUMN_A、COLUMN_C、COLUMN_D”在抽取数据查询数据的 select 子句即为 select COLUMN_A， COLUMN_C，COLUMN_D from XXX”。SYNC_QUERY 字段的设计后还可以实现一些其他的好处，例如我们抽取出的数据的字段需要按照一定的顺序，例如，表字段顺序 COLUMN_A、COLUMN_B、COLUMN_C 但我们希望保存的字段顺序为 COLUMN_A、COLUMN_C、COLUMN_B，或者我们需要先对某个字段进行特殊处理再抽取出来，如 COLUMN_A，FUNC(COLUMN_B)，COLUMN_C）都可以通过对 SYNC_QUERY 字段的值进行配置来实现。

SYNC_ADDDATE

配置的添加时间。表示添加配置的时间，没有特殊含义。

数据库数据抽取流程如下：

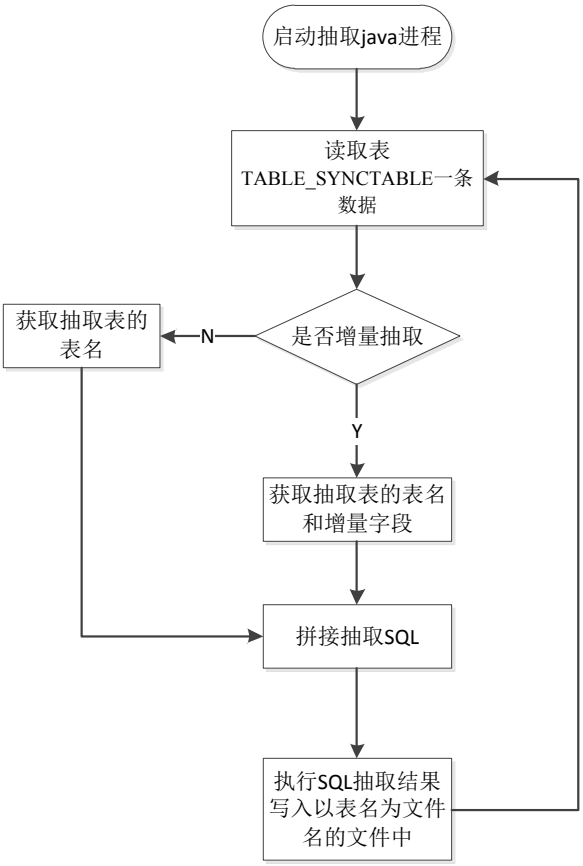

图 4-1 数据库抽取数据流程图

Linux 下使用 crontab 建立定时任务调用上述 Java 程序，每天定时抽取数据，然后打包 SCP 到 Hadoop 服务器。

每天调用定时任务主角本

```
10 00 * * * sh sync_tables.sh
```

4.1.3 数据导入

数据导入一种是 Flume 直接写入 HDFS，对于不同的 sink 创建不同的文件路径进行接收。另一种是把文件放入 HDFS 或者 Hive 表中。

数据导入

```
#导入到HDFS 中
hadoop fs -put TABLE_PI_LOGINFO_${TODAY}.log
/user/hadoop/TABLE_PI_LOGINFO
#导入到hive 表中
load data local inpath '/home/hadoop/TABLE_PI_LOGINFO_${TODAY}.log'
overwrite into table    TABLE_PI_LOGINFO partition(dt='${TODAY}')
```

4.2 数据预处理模块

数据的预处理主要是为了除去数据中的不协调因素，删除不合法的数据，使

得数据规范化，在以后的数据分析中提供高质量的数据，便于高效，准确的分析数据。

4.2.1 数据清洗

数据清洗是为了识别和删除数据中的“脏数据”。数据来源于多个应用服务器和数据服务器，服务器每天要处理数以亿计的请求，可能会发生一些错误，有些数据的字段是我们分析数据所必须的，如果这些字段的数据丢失，那么这些数据在分析的时候就失去了意义，我们把这些数据称之为“脏数据”。同时有些数据并不是我们数据分析的时候所关心的数据，这种数据字段出现错误情况就可以忽略。数据清洗就是要去掉那些必须的数据字段中不完整的数据。导致数据记录不完整的原因是多种多样的，比如用户在联网的时候突然断网了，导致某些数据记录不完整，或者根本无法获取到用户的信息。

数据清洗主要清理残缺数据、错误数据、重复数据三种情况。

残缺数据包括有效数据和无效数据，有效残缺数据是指缺失的数据项是进行数据分析时关系不大或者毫无关系的数据项。例如：分析用户活跃信息时，我们主要分析用户联网类型，联网时间，产品信息等，而对于用户联网的内容我们不关心。这种数据记为有效数据。但是如果用户信息缺失了，那么数据就为无效数据了。数据清洗主要是删除无效数据。

错误数据表示某些数据项由于特殊原因导致记录出错。错误数据也包括有效数据和无效数据两种。

重复数据有两种情况，一种是指在有唯一标识的字段中又出现了相同的值。这种重复数据主要是因为用户数据的变更。另一种是由于数据的冗余，例如，用户一天有多条联网记录，我们分析用户活跃度时只关心用户是否联网，所以我们只保留每天最后一条联网记录即可。处理流程如下：

图 4-2 联网数据去重流程图

4.2.2 数据集成

由于数据源是多种多样的，导致收集到的数据格式也是多种多样的，有 Log4j 记录的格式化数据，有数据库数据，也有来自于 Web 页面的数据。我们需要把各种各样的数据进行统一格式，集成到一起，这就应用到了数据集成技术。数据集成是把不同格式、性质、来源的数据在有机地集中在一起，提供相对集中，易于处理的共享数据。数据集成技术解决了数据分散的问题，既保证了数据的完整性，又可以做到数据最小的冗余。

本系统中，我们主要针对来自于 Web 页面的记录的 JSON 格式的数据进行集成。普通的 Log4j 记录的日志和数据库数据我们在数据记录的时候就做了规范化处理，这些数据格式都是规范化的，我们无需再进行数据格式处理，只需要把 JSON 格式的数据处理就行。JSON(JavaScript Object Notation)[34] 是一种轻量级的数据交换格式。非常适合服务器与 JavaScript 进行交互。 JSON 有两种结构，

对象和数组。对象：在 js 中对象是以“{}”括起来的，数据结构为{key：value，key：value，……}的键值对的结构。在面向对象的语言中，key 为对象的属性，value 为对应的属性值。数组：在 js 中数组是中括号“[]”括起来的，数据结构为 [“1”，“2”，“3”，……]，取值方式是使用索引获取数组内的值，字段值的类型不固定，可以是数字、字符串、数组、对象等等。

本系统中 JSON 格式数据都是以键值对的对象数据存在的，我们需要把他解析为表结构的数据格式。处理过程如下：

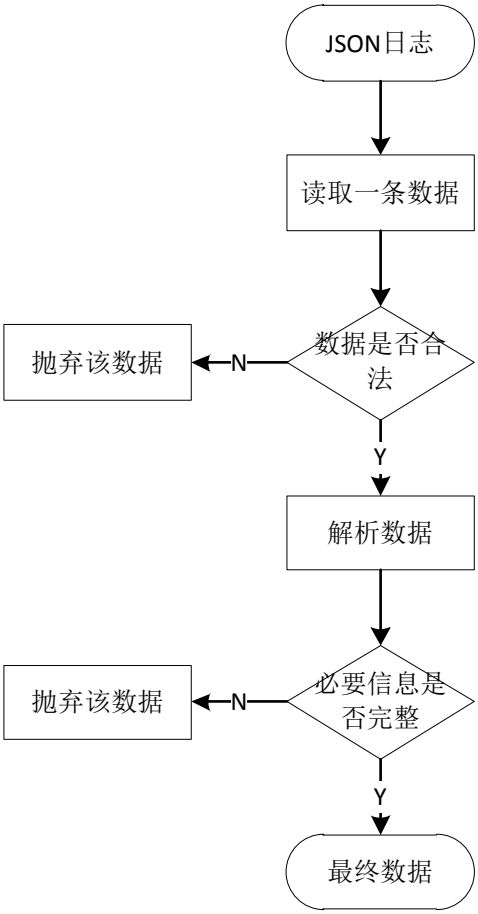

图 4-3 数据集成流程图

JSON 的这种存储数据的形式在非结构化的数据中有很好的灵活性，存储数据的效率也很高，但是在分析数据时候数据的冗余很大，需要消耗很大的传输带宽和磁盘空间。因此，对数据进行预处理可以大大减少数据的冗余，提高数据传输和处理效率。

表 4-2 预处理数据和原始数据对比

	JSON 格式	规范化格式
数据条数	1875465340	1875465340
大小	900G	210G
传输时间	25137s	5681s

4.2.3 用户数据初始化

数据分析人员在分析数据的时候一般都会结合用户去分析，用户是一个产品生存的根本，例如，分析用户的活跃度，分析用户的联网信息等。用户的基本信息是分析人员查询频率较高的数据，相对于其他数据信息，用户基本信息都比较简单，占用空间较小。一个用户在使用产品的时候会产生大量的数据信息，但是在整个生命周期内，用户的基本信息往往是不会发生变化的。用户的一些信息也不是存储在同一个数据表中，为了提高查询效率，我们预先把用户的基本信息都整合到一起，在后续的处理中只需要关联用户基本信息就行，不必要再去关联大量的表。

用户基本信息是常用数据，不必要在每次查询的时候再加载到内存，我们可以让用户基本数据常驻内存。我们把用户基本信息存储在 Redis 中。Redis 加载逻辑如下：

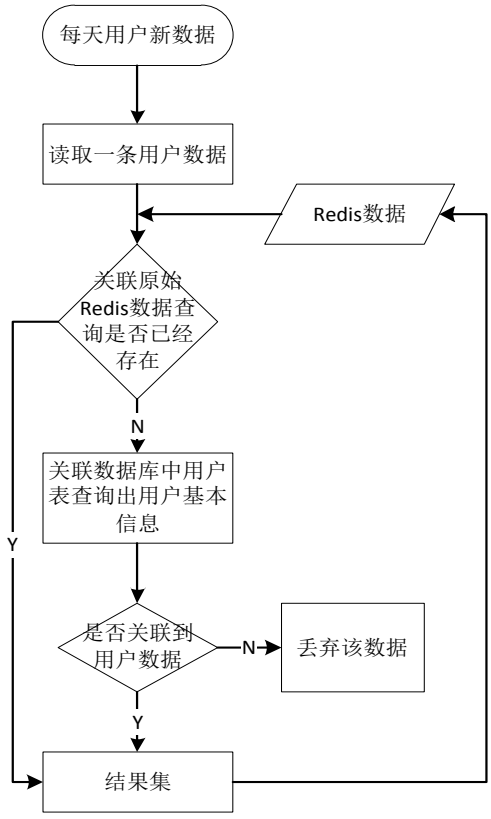

图 4-3 Redis 加载数据流程图

每天需要把前一天的用户数据加载到 Redis 中，首先读取一条数据，关联 Redis 看是否该用户信息已经加载，如果已加载则忽略，若没有该用户信息，则关联数据库中用户信息相关的表，找出新用户的基本属性信息，加载到 Redis 中。Redis 采用以用户唯一 ID 作为 key，其他属性信息作为值来保存用户数据。Redis 中的用户数据经过了用户信息初始化的过程，数据都比较规范，便于下一步的数据分析。

数据初始化环节其实就是对热数据进行一次整合，虽然增加了数据的冗余，但是在以后的分析中省去了很多的数据关联操作，大大提高了数据分析的性能。

在经过了数据预处理的各个过程之后，数据对于用户分析需求就更加明确了，丢弃掉了数据中包含的“脏数据”，剩下的数据都是可以直接进行数据分析的，通过数据集成把来自多个数据源的数据集成起来，把数据分析需要的热数据加载到 Redis 内存数据库中，用层次化的概念对源数据进行了封装，更加便于用户进行分析需求。

数据预处理环节保证了数据的完整，数据分析的高效，是整个系统不可或缺的环节。

4.3 数据分析模块

数据分析模块主要负责数据统计分析，根据分析人员的需求，制定分析策略，进行数据分析。本节主要根据常用的数据分析模型来讲述数据分析的相关处理办法。

4.3.1 Hive 数据分析

Hive 处理方式是使用 HiveQL 语言进行数据分析，HiveQL 语言和数据库 SQL 语言类似，便于操作。HiveQL 语句会转换为 MapReduce 程序进行执行。例如：统计所有用户数只需要在 hive 接口中执行 select count(user) from table_users，Hive 表数据的关联也和普通 SQL 类似，如：select userid,product from table_users tu,table_product tp where tu.prcid_fk=tp.prc_id。Hive 语言进行数据分析方便快捷，不需要开发 MapReduce 程序，减少了分析人员的工作量。

在进行 Hive 分析中，我们遇到了一个问题。在进行某些分组统计或者关联查询的时候，数据分析效率特别低。

数据倾斜[35]是指，MapReduce 程序执行时，Reduce 节点大部分执行完毕，但是有一个或者几个 Reduce 节点运行很慢，导致整个程序的处理时间很长，这是因为某一个 key 的条数比其他 key 多很多（有时是百倍或者千倍之多），这条 key 所在的 Reduce 节点所处理的数据量比其他节点就大很多，从而导致某一个或者几个节点迟迟运行不完。比如，我们分国家统计用户数的时候，有的国家用户比较少，有的国家很多，这样就导致用户多的国家执行效率很慢，从而影响整个数据分析的效率。

导致数据倾斜主要有以下几种情况：

- 表之间的关联操作，其中一个表较小，但是关联的 Key 值比较集中，这样导致分发到一个或某几个 Reduce 上的数据远远大于分发到其他 Reduce 的数据。数据量少的执行完成之后一直要等待，等全部执行完成

之后进程结束。还有就是大表与大表进行关联，但是关联字段的 0 值或空值比较多，导致所有空值或 0 值都分发给一个 Reduce 进行执行，也导致分析效率很慢。

- Group by 分组查询。按照某一个维度分组时，某一个值数据过多，导致 Reduce 处理该值的时候很慢。
- Count distinct 查询的某一数据重复值过多，导致 Reduce 很慢。

数据倾斜的直观表现就是数据查询的时候任务进度长时间维持 99%或接近 100%的一个值，查看任务监控页面可以发现有一个或者几个 Reduce 子进程未完成。因为处理的数据量过大。

针对数据倾斜的解决方案：

1. 参数调整。

（1） hive.map.aggr=true

数据在 Map 进行聚合，相当于中间经过 Combiner 函数进行预处理，Combiner 可以看作是 Reducer 的助手，它能够减少 Mapper 的输出，最终降低 Reducer 端的负载。但是 Combine 的使用也有限制，他并不会适合所有的数据分析场景，以下是一些适合的场景：对数据进行汇总统计，比如求和；取最大最小值的场景，只保留 Mapper 的最大最小值，不会改变最后的最大最小值结果；对于求平均数的场景，Combine 就不适合。

Combine 函数在 Map-Reduce 模型中是可选的。在这里为了减少网络传输和整体程序的性能，使用 Combiner。Combiner 接收一个 Mapper 处理的结果键值对，把他们合并成一个新的键值对。可以用如下过程来表示：

Map(Key1，Value1) → Combine(Key2，Value2)

这里，Key1，Value1 对应 Mapper 端输出的键和值，对于同一个 Key1，把其相应的 Value1 的内容求和，得到 Value2，Key2 的内容实际上和 Key1 是一样的，新的键值对（Key2，Value2）作为输入传输到 Reduce 函数中。

Combiner 类在实现的时候需要继承 Reducer 类，重写其中的 Reduce 方法，对 Mapper 的输出，重新生成键值对作为 Reduce 的输入，下面是 Combiner 函数的具体处理逻辑：

Combine 函数实现

```
public static class CombinerCount
    extends Reducer<Text，IntWritable，Text，IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key,Iterable<IntWritable> values,Context context
            ) throws IOException， InterruptedException {
                int sum = 0;
                for (IntWritable val : values) {
                   sum += val.get();
                }
                result.set(sum);
                context.write(key， result);
            }
        }
```

（2） hive.groupby.skewindata=true

这个参数是当数据进行分组查询而产生数据倾斜的时候进行负载均衡处理。当该参数设定为 true，查询计划中就会生成两个 MapReduce Job。在第一个 MapReduce Job 中，Map 的输出结果会随机的分发到不同的 Reduce 中，每个 Reduce 做部分聚合操作，这样处理的后果是相同的分组 Key 有可能被分发到不同的 Reduce 中，这样可以做到负载均衡的效果。第二个 MapReduce Job 再根据第一个 MapReduce Job 输出的结果按照分组 Key 分布到 Reduce 中，这个过程可以保证相同的分组 Key 可以分发到同一个 Reduce 中，最后完成最终的聚合操作。

Hive 参数调整主要针对于数据键值对不均衡的情况下，通过 Reduce 端的负载均衡处理来避免数据倾斜提高数据处理效率。

2. 查询语句调整。

表间的 join 关联查询，选择合适的驱动表，选择 join Key 分布比较均匀的表作为驱动表；提前对表进行过滤，过滤掉与结果无关的数据，两个表 join 的时候数据量达到尽可能的小；使用 map join 小的维度表先进入内存，在 map 端进行关联；空值 key 随机加上一个数，把空值数据分配到不同的 Reduce 上，由于空值是关联不上的，所以不会影响数据的结果。

4.3.2 MapReduce 数据分析

Hive 只提供了一些简单了统计功能，由于业务需要，有些统计需求不是简单的关联、分组就能统计出来的，要想实现更复杂功能，我们需要自己开发一些用户自定义函数（UDF）或者开发 MapReduce 程序。用户自定义函数是 Hive 提

供的 MapReduce 接口，定义函数的时候必须先继承 UDF 类。重写其中的 evaluate 函数。下面简单介绍一下根据需求开发的一个 UDF 函数，主要实现的是根据某个字段分组，给数据进行 rowid 标记。

自定义函数的实现

```
import org.apache.hadoop.hive.ql.exec.UDF;
public class RowNumber extends UDF {
    private static int MAX_VALUE = 50;
    private static String comparedColumn[] = new String[MAX_VALUE];
    private static int rowNum = 1;
    public int evaluate(Object... args) {
        String columnValue[] = new String[args.length];
        for (int i = 0; i < args.length; i++) {
            columnValue[i] = args[i].toString();
        }
        if (rowNum == 1) {
            for (int i = 0; i < columnValue.length; i++)
                comparedColumn[i] = columnValue[i];
        }
        for (int i = 0; i < columnValue.length; i++) {
            if (!comparedColumn[i].equals(columnValue[i])) {
                for (int j = 0; j < columnValue.length; j++) {
                    comparedColumn[j] = columnValue[j];
                }
                rowNum = 1;
                return rowNum++;
            }
        }
        return rowNum++;
    }
}
```

上述函数主要实现了类似于 Oracle 数据库函数的 rank()over()的功能，根据一个字段进行数据分组，根据组内一个字段排序然后对数据进行 rowid 标记，最终根据标记位获取需求数据的函数。用户自定义函数可以根据用户需求具体实现，没有特殊的限制。

4.4 监控告警模块

监控模块负责数据在传输过程中的完整性和一致性进行监督，同时监控着各个服务器节点的运行状况，保证系统准确、高效的运行。

4.4.1 数据流监控

数据流部分监控主要是数据在各个环节中进行着数据传输，在数据传输过程中我们要保证数据的完整性。数据监控主要为数据量的对比，数据处理前后数据量变化情况，今天与昨天数据量变化情况的对比。告警方式主要为邮件方式。流程如下：

首先，脚本程序读取当天的数据文件获取初始的数据总量。然后，数据写入 HDFS 或者 Hive 表中。写入完成后脚本读取完成的 HDFS 文件或当天插入 Hive 表中的数据量。最后脚本获取前一天的数据量，三种数据进行对比，查看数据差别是否在阈值范围之内，超出阈值发送告警文件。

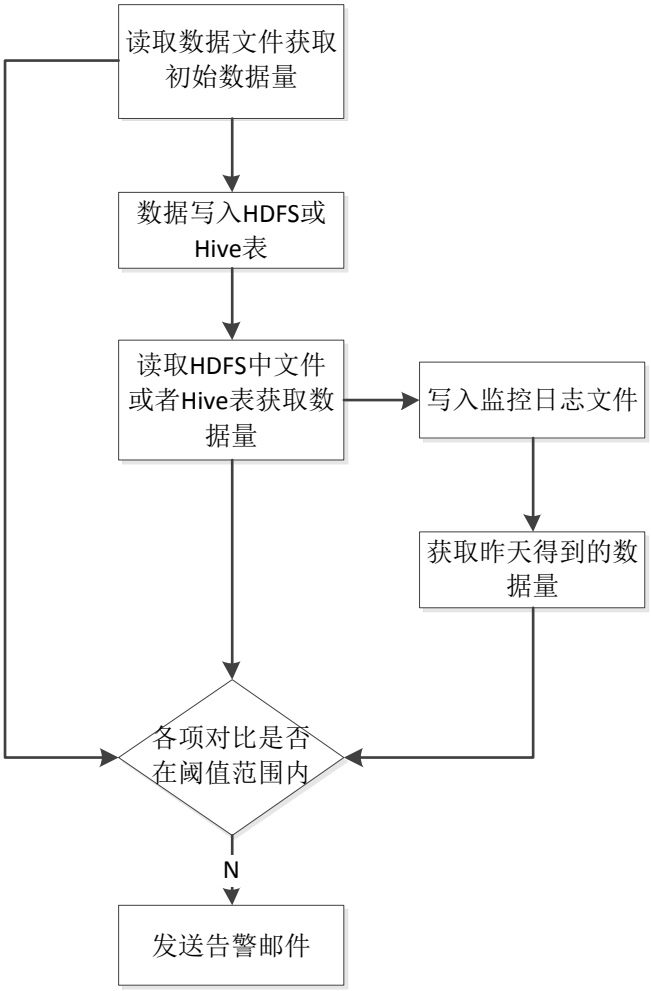

图 4-3 数据流监控流程图

4.4.2 数据处理监控

数据处理监控主要监控数据在处理的过程中是否发生异常情况导致数据处理失败。监控数据处理的每个环节，记录每个环节数据处理情况。发送处理情况邮件。

4.4.3 数据粒度监控

数据粒度监控主要是针对实时数据进行监控，根据监控规则，分析数据是否在阈值范围之内。主要流程如下：

首先，进入实时监控的入口，读取数据库中的监控粒度加载到内存中，然后结合监控粒度项分析实时日志，得到的分析结果和配置的公共变量阈值进行对比，分析是否超出配置的阈值，返回监控结果，如果超出阈值则发送具体数据错误邮件，否则不发邮件。

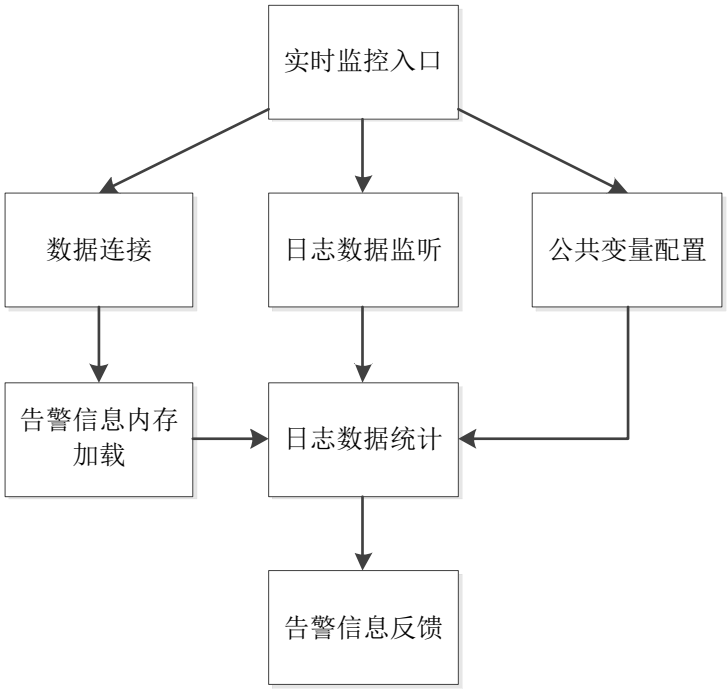

图 4-4 数据粒度监控流程图

4.4.4 服务器监控

Hadoop 提供了很多的监控工具的接口，本系统使用 Ganglia 工具。Ganglia 配置起来非常简单，只需要配置$HADOOP_HOME/conf/hadoop-metrics.properties 文件即可，配置完成后，把文件负责到各个节点，然后重启 Hadoop 服务。

同时我们也采取 shell 脚本监控并存的方式，shell 脚本也定时读取服务器的一些信息，如果超出阈值则发送告警邮件。

4.5 数据展示模块

数据展示模块是用户和服务器连接的纽带，用户将自己的需求输入，系统将

结果反馈给用户。所以，展现模块要具有清晰明了的特点，让用户很清楚的知道功能，输入需求与得到结果。

数据处理结果依旧保存在 HDFS 中，我们为了便于展示，把数据结果放入 Hive 表或者关系型数据库中。Hive 表中保存结果集大的数据，关系型数据库保存结果集较小的数据。使用 Sqoop 工具把 HDFS 中的数据导入到数据库中，数据库我们使用的是 Oracle 数据库。

首先我们在 oracle 数据库中创建对应的表。

创建表的 SQL

```
Create table table_result
(
 Result_date date，
 Result_user varchar2(20)，
 Result_count number
)
```

然后把数据导入到 Oracle 数据库中。

Sqoop 导入 Oracle 脚本

```
sqoop export --connect jdbc:oracle:thin:@//IP:PORT:orcl --username USER
--password PASSWORD --table tbl_test --export-dir /user/result/
--input-fields-terminated-by '\t'
```

4.5.1 搭建 Web 服务

数据分析的结果最终还是要友好的展示给数据分析人员。这就需要搭建一套 Web 服务系统，Web 系统是用户和数据交互的唯一途径，是用户使用的平台。Web 系统系统采用 J2EE 架构，应用 Spring 的 MVC 模式进行设计和实现。

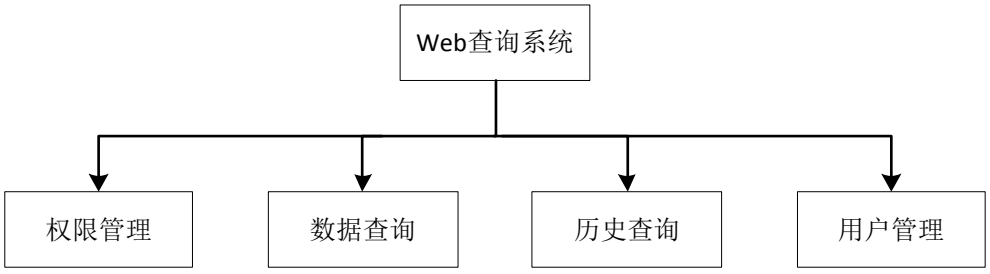

图 4-5 Web 服务架构图

从上图中可以看出 Web 页面主要实现权限管理、数据查询、历史查询、用户管理四个功能部分。

权限管理在下一节中具体阐述。

数据查询包括 Hive 表数据的查询和报表数据查询两方面。历史查询主要是

历史记录的保存功能，用户可以查询自己以前查询的记录。用户管理主要包括用户的增、删以及用户角色的分配、权限的设置等。

4.5.2 权限设置

报表权限，数据库中一张表是存储用户权限信息的，每个报表有一个属于自己的 ID，配置表中保存这用户可以查询的报表的 ID，页面展示的时候根据登陆用户的权限显示出报表的列表，用户选择查询的报表显示报表信息。

1. 表权限

用户可以自己写 HiveQL 语句查询 HIVE 表中的数据。用户权限表保存这用户可以查询的表的信息，根据用户输入进行解析判断用户是否有权限查询所查的表。

2. 维度权限

用户针对于某一个维度并不是所有的数据都有权限查询，为了保证数据的安全，需要控制用户所查维度权限。例如，国家维度，用户 A 可以查询所有数据，用户 B 只可以查询中国的数据，用户 C 可以查询美国的数据。这样我们在查询关于国家的数据的时候就需要控制国家维度的权限。有一个表是存储用户可查询的维度具体值，当用户查询相关维度的时候就关联该表，只查询出用户有权限的数据。类似于 Oracle 的 VPD[36]权限控制。具体实现步骤如下：

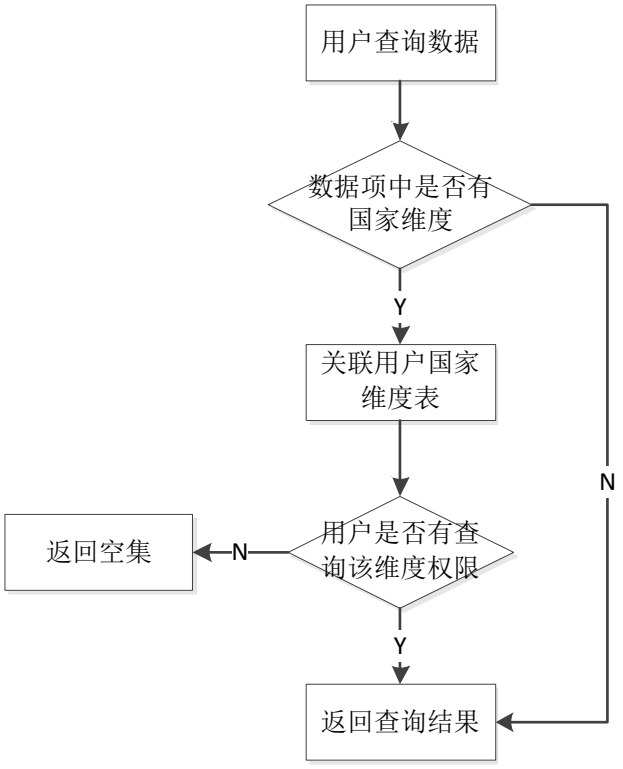

图 4-6 权限控制流程图

4.6 本章小结

本章主要介绍了本系统中各个模块详细设计和具体实现，包括数据收集、数据预处理、数据分析、监控和数据展示五个模块。讲述了各个模块功能的设计以及实现，完整的构建了整个基于 Hadoop 的海量业务数据分析平台，并且搭建了 Web 应用服务把数据友好的展示出来。

第五章 系统部署与测试

前面已经把系统设计和实现都详细的介绍了，本章主要讲述系统环境的搭建以及系统的测试与数据展示。系统测试部分主要负责验证数据的准确性和计算效率，在各个场景下进行系统的验证。同时展示了 Web 页面的使用效果。

5.1 环境搭建

本系统的正式环境下使用的 Hadoop 集群共有 30 个节点，其中一台服务器作为 Master 节点，HDFS 的 NameNode 节点和 MapReduce 模型的 Jobtracker 都部署在上面，一台服务器作为 Secondary NameNode 节点，Secondary NameNode 节点用来备份 Master 节点的数据，防止 Master 节点出现故障。其他 28 台服务器为 Slaver 节点，用来保存 HDFS 数据和执行计算任务。

表 5-1 IP 列表及主机名

IP	主机名
192.168.0.18	hadoop.master
192.168.0.19	hadoop.second
192.168.0.20	hadoop20
192.168.0.21	hadoop21
192.168.0.22	hadoop22
192.168.0.23	hadoop23
192.168.0.24	hadoop24
192.168.0.25	hadoop25
192.168.0.26	hadoop26
192.168.0.27	hadoop27
192.168.1.151	hadoop151
192.168.1.152	hadoop152
192.168.1.153	hadoop153
192.168.1.154	hadoop154
192.168.1.155	hadoop155
192.168.1.156	hadoop156
192.168.1.157	hadoop157
192.168.1.158	hadoop158
192.168.1.159	hadoop159
192.168.1.160	hadoop160
192.168.1.161	hadoop161
192.168.1.162	hadoop162

表 5-2 IP 列表及主机名（续上表）

IP	主机名
192.168.1.163	hadoop163
192.168.1.164	hadoop164
192.168.1.165	hadoop165
192.168.1.166	hadoop166
192.168.1.167	hadoop167
192.168.1.168	hadoop168
192.168.1.169	hadoop169
192.168.1.170	hadoop170

首先，需要配置 SSH 服务，Hadoop 集群中的主节点和从节点服务器之间都是无密钥访问。首先在主节点上生成 SSH 受信证书，然后复制到从节点，则主节点不需要密码就可以访问从节点。生成证书命令为：ssh-keygen -t rsa。

把产生的 id_rsa.pub 中的内容复制到 authorized_keys 文件中，然后把生成的 authorized_keys 文件复制到 Master 节点需要访问的其他节点的/user/home/.ssh 目录下，这样就对于当前用户无需密码就可以访问其他机器。其中，对于 authorized_keys 文件的权限必须设置为 600，其他用户不能对此文件有读写权限，这样才能相互无密码访问。

其次，需要配置 hosts 文件，hosts 文件的作用是解析主机名和 IP 地址的对应关系。对于 NameNode 节点则需要在其 hosts 文件中添加集群中所有节点的 IP 地址及对应的主机名，对于 DataNode 节点，则至少需要在 hosts 文件中添加 NameNode 节点和本机的 IP 地址及对应的主机名，为了方便集群中计算机之间的通信，DataNode 节点的 hosts 文件中也加入其他节点的信息。例如下面 hosts 文件的部分配置。

Host 配置

192.168.0.18　hadoop.master

192.168.0.20　hadoop20

192.168.0.21　hadoop21

……

5.1.1 Hadoop 配置

Hadoop 安装以前需要在集群中所有的节点上建立相同的用户，并以该用户的 home 目录作为 Hadoop 的安装目录。本系统中安装 Hadoop 的用户名就是 hadoop。各个节点的安装文件和配置文件都是相同的。所以，我们先安装好 NameNode 节点上的 Hadoop，然后将安装文件和配置文件复制到其他各个节点即可。安装好后需要对 Hadoop 集群进行配置。

修改 Hadoop 目录下/conf 文件夹中的配置文件，需要修改的配置文件有：

master、slaves、core-site.xml、hdfs-site.xml 和 mapred-site.xml。

master 文件添加 Namenode 节点主机名，slaves 文件添加所有 Datanode 节点的主机名。Master 配置如下：

Master/Slavers 配置

```
Master
hadoop.master

Slaves
hadoop20
hadoop21
hadoop22
……
```

文件 core-site.xml 配置 Namenode 的 IP 地址，端口号。通过 IP 地址和端口号，我们可以在 Web 端进行查看任务的运行状况和各个节点的情况。

core-site.xml 配置

```
<configuration>
    <property>
            <name>fs.default.name</name>
            <value>hdfs://hadoop81:8020</value>
    </property>
</configuration>
```

文件 hdfs-site.xml 配置 HDFS 文件数据保存的物理位置，命名空间数据保存位置，以及通过 Web 访问查看的 IP 和端口号，数据冗余几份，文件是否允许增量插入等等。

hdfs-site.xml 配置

```
<configuration>
    <property>
            <name>dfs.name.dir</name>
            <value>/opt1/data/name</value>
    </property>
    <property>
            <name>dfs.data.dir</name>
            <value>/opt1/data/data</value>
    </property>
```

```
    <property>
        <name>dfs.http.address</name>
        <value>hadoop81:50070</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <property>
        <name>dfs.support.append</name>
        <value>true</value>
    </property>
</configuration>
```

文件 mapred-site.xml 主要配置 JobTracker 的相关信息，如访问 IP 及端口号，通过 Web 方式可以访问 JobTracker 相关信息。

mapred-site.xml 配置

```
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>hadoop81:9001</value>
    </property>
</configuration>
```

配置完成之后把配置的文件复制到 namenode 节点的相应位置。启动 Hadoop 服务，$HADOOP_HOME/bin/start-all.sh。

5.1.2 Hive 配置

Hive 需要一个数据库作为保存 Hive 基本信息的地方。我们使用的是 Mysql 数据库。数据库中保存的是 Hive 的元数据，包括表的字段信息、索引信息、数据保存物理位置等等一系列的整个 Hive 的元数据。配置 Hive 主要配置一个 hive-site.xml 文件即可。主要配置有 Mysql 数据库的连接 URL 和数据库名，数据库的连接驱动，用户名及密码等相关信息。

hive-site.xml 配置

```
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://hadoop81/hivedb</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hiveuser</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>hadoop</value>
    </property>
    <property>
        <name>datanucleus.autoCreateSchema</name>
        <value>false</value>
    </property>
    <property>
        <name>datanucleus.fixedDatastore</name>
        <value>true</value>
    </property>
</configuration>
```

5.2 分析测试

5.2.1 数据分组统计测试

测试方式为关系型数据库与 Hadoop 集群在数据统计方面的效率对比。两种统计中数据都是完全一致的。通过控制数据量，来对比 Hadoop 分布式处理与数据库单机处理的性能情况。

其中，使用的关系型数据库服务器配 CPU 为 8 核 Intel(R) 2.80GHz CPU，内

存容量 64G，性能非常好。数据格式为用户联网信息的情况。每条数据分为 10 个字段，包括用户信息，时间信息，产品信息，联网类型，联网内容，以及状态信息等。

测试需求，按天进行分组，分组统计出每天联网用户数，用户数需要去重。

测试结果如下：

表 5-3 聚合测试统计结果

数据行数（万行）	数据文件大小（MB）	关系型数据库执行时间	Hadoop 执行时间
100	1562	2.36s	35s
1000	16258	79s	58s
2000	33580	259s	89s
5000	87690	589s	103s

从实验结果可以看出，当数据量较小的时候关系型数据库的处理时间要优于 Hadoop 分布式，因为 Hadoop 需要一定的初始化时间，同时还存在着数据的传输，所以数据量小时，这些时间加起来肯定不如数据库效率高。但是随着数据量的增加，Hadoop 分布式计算的优势就体现出来了。所以 Hadoop 更适合于大数据量的分析。

有一个产品用户数很多，每天联网数据量很大，其他产品数据量相对较小，在使用 Hadoop 进行分布式查询的时候产生了数据倾斜。下面我们分析一下数据倾斜情况下的优化。控制数据量，分析不同数据量情况下的对比。

表 5-4 数据倾斜优化对比

数据量（万行）	数据文件大小（MB）	Hadoop 执行时间	Hadoop 执行时间 hive.groupby.skewindata=true
100	2819	216s	156s
1000	29681	267s	195s
2000	60568	352s	321s
5000	156890	521s	468s

通过上面表格可以看出，在数据分布不均匀的情况下，进行分组查询，我们发现有一个或几个 Reduce 比其他 Reduce 执行要慢很多很多。因为大部分的数据都是由这几个 Reduce 来计算，所以我们进行了优化，使得有一个中间函数 Combine 进行数据过滤，执行效率有了明显提高。但是当数据分布均匀的时候我们使用 Combine 反而会变得较慢，因为数据进行了两次 MapReduce 操作，延长了数据处理时间。

5.2.2 数据关联统计对比

分析数据在 hive.map.aggr 参数为 true 的情况下的 Map 端关联和正常情况下 Reduce 端关联的情况。

表 5-5 Map 端关联和 Reduce 端关联对比

Left 表大小	表行数（万）	Right 表大小	表行数（万）	Map 端关联	Reduce 端关联
15M	1	33G	2000	80s	359s
2.6G	2100	33G	2000	323s	534s
11G	9820	33G	2000		1029s
22G	19640	33G	2000		1632s

在测试过程中，当进行 Map 端关联测试的时候，当数据大于 2.6G 时，测试程序会发生 java.lang.OutOfMemoryError: GC overhead limit exceeded。这是由于任务所使用的内存超过了 jvm 所能分配的最大内存量，Map 端的小表无法放入内存中。

从结果可以看出，当其中一个表较小时，使用 Map 端 join 的方式效率最高，随着数据量的增加，我们的内存无法满足把一个表的数据放入的时候 Reduce 端的 join 成了我们的选择。在数据关联操作时根据不同的需求来进行不同的关联方式，必要的时候我们还需要自己开发 MapReduce 程序来处理适合业务需求的数据分析。

5.3 数据展示

数据展示时通过 Web 页面的方式展现出来。主要分为用户自定义查询系统和报表系统两个平台。

5.3.1 用户自定义查询系统

用户在浏览器端输入访问该系统的网址，进入系统首页，输入用户名密码进行登录。登录后可以进行自助查询服务。如下图所示：

图 5-1 自助查询页面

或者我们也可以自己写 HiveQL，进行查询。

图 5-2 自助查询 HiveQL 页面

查询结果可以进行查看，也可以下载到本地。

图 5-3 查询结果页面

同时可以查询历史记录，查看历史查询结果，也可以把结果下载到本地文件系统。

图 5-4 历史查询记录页面

可以查看自己有哪些权限表，表的字段信息。

图 5-5 Hive 表信息

还有其他一些功能，如上传数据到 HDFS，查询 HDFS 中的文件，个人信息关联等等。

5.3.2 报表系统

报表系统是根据已知需求预先把基本数据处理出来，用户需要查询报表直接选择所需报表即可，这样大大减少了用户查询等待时间。

报表系统登录之后，可以看到自己有哪些角色，可以查询哪些报表数据。

图 5-6 用户权限页面

点击进入具体查询报表，会出现报表中各个数据项，根据需求选择所需要或者限制的数据项，然后点击提交按钮。

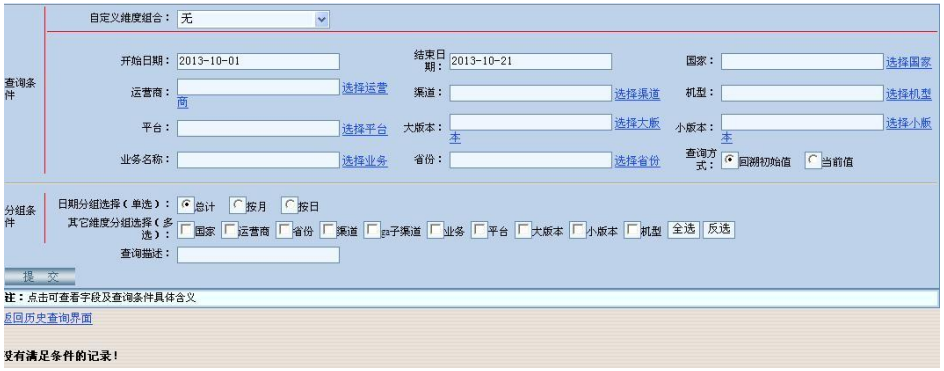

图 5-7 报表查询页面

查询结果会返回到页面，用户可以查看也可以进行下载。

图 5-8 报表查询结果

也可以查询历史查询记录。

图 5-9 报表查询历史记录

5.4 本章小结

本章主要讲述了硬件环境的搭建和数据验证测试以及 Web 查询系统的展示。环境搭建主要包括服务器的配置，Hadoop 安装配置和 Hive 的安装配置。数据测试主要是对分组数据统计和数据关联统计进行了测试，并达到了预期效果。通过使用 Web 查询系统使得数据查询需求更加简便快捷，目前系统已经正式投入使用，效果很好。

第六章 总结与展望

6.1 本文工作总结

本文讲述了一个基于 Hadoop 的海量业务数据分析的设计和实现的方案。该系统应用了 MapReduce 模型进行数据分析，采用 HDFS 分布式文件系统进行数据存储。使用 HiveQL 语言进行数据查询，用户能够通过简单的操作就可以进行产品数据分析，挖掘出有用的数据。

本系统共分为五个模块，数据收集模块，数据预处理模块，数据分析模块，监控模块，数据展示模块。数据收集模块主要收集各个服务器的数据，包括日志数据和数据库数据，日志数据通过 Flume 进行数据的实时收集，和 SCP 复制的方式隔天收集。数据库数据采用当天抽取前一天的数据的方式进行隔天数据同步。最终把数据放入到 HDFS 文件系统中。这样根据数据紧急程度分开收集解决了数据在传输过程中的负载均衡和网络拥堵问题。数据预处理模块主要负责处理数据中的不协调因素，包括数据清洗，数据集成以及对用户类的热数据加载到 Redis，以便在后续的处理过程中减少磁盘读取，大大提高数据处理效率。数据分析模块主要分析用户对产品的使用情况，新增用户数，用户活跃度，产品受欢迎程度等等。分析方法主要采用 HiveQL 语言结合特定的 MapReduce 程序，主要进行数据分组统计以及多数据源的关联分析。分布式的并行计算较传统的数据库数据分析相比，具有高性能，高吞吐量的特点。监控模块对数据在各个环节的流动中进行实时监控，保证数据的准确性，同时也监控着各个节点服务器，保证系统的正常运行。数据展示模块把数据友好的展示给用户，构建 Web 服务，用户可直接通过 Web 的方式访问数据处理结果，用户可以直接查询结果报表也可以通过写 HiveQL 语句直接查询 Hive 表中的数据。同时，本系统有良好的权限管理，用户对于报表的权限，Hive 表的权限，乃至具体数据维度的权限都做了明确的设置，非常适合企业的需求。

经过实际数据验证，本系统相对于传统关系型数据库，极大体现了 Hadoop 分布式计算的优势，对于处理大数据有明显的优势。系统部署简单，可扩展性好，随时根据需要扩充节点。

6.2 今后工作展望

本系统的研发对企业来说具有重要的意义，随着互联网的发展，人们对网络的依赖越来越大，企业的数据量也成倍的增长，对数据分析的需求也不断增加。

本系统很好的解决了对于大数据分析的问题。但是，系统中也还存在着一些问题需要改进。

首先，系统的功能还有待完善，对于过于复杂的数据分析还不具备，仅仅处于基础层面。其次，数据收集中 Flume 和 Log4j 的兼容性还存在问题，Log4j 断掉重启之后 Flume 进程不会自动重启，Log4j 可以写数据，但接收端接收不到，导致数据丢失，现在处理办法是在 Log4j 中有个进程来监听 Flume，但不完善，仍需要改进。第三，服务器故障检测存在延时报告，导致不能及时发现处理，后续工作需要解决。最后，数据细节监控不完善，只是针对数据量进行监控，后续需要解决计算时间延迟问题的监控。

随着系统的不断改进，它的功能更加完善、数据一致性更有保障，处理的效率将更高、更可靠。

参考文献

[1] 姜奇平. 大数据时代到来[J].互联网周刊,2012,第 2 期:5-6.

[2] 朱明. 数据挖掘[M]. 中国科学技术大学出版社, 2008.

[3] 史忠植. 知识发现[M]. 清华大学出版社, 2002.

[4] 杨德仁. 关系型数据库[J]. 程序员, 2002 (7).

[5] Edgar Frank Codd, “A Relational Model of Data for Large Shared Data Banks”, Communications of the ACM, June,1970,377-387.

[6] 梁冰,陈丹丹,苏宇. SQl 语言参考大全[M]. 人民邮电出版社, 2008.

[7] 陈京民. 数据仓库与数据挖掘技术[M]. 电子工业出版社, 2007.

[8] http://wiki.e-works.net.cn/wikipage/200811/entry2286.htm.

[9] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, “The Google File System”, SOSP’03, October 19–22, 2003.

[10]Jeffrey Dean,Sanjay Ghemawat, “MapReduce: Simpli ed Data Processing on Large Clusters”, OSDI, 2004.

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, “Bigtable: A Distributed Storage System for Structured Data”, OSDI, 2006.

[12]White T. Hadoop: the definitive guide[M]. O'Reilly, 2012.

[13] http://hadoop.apache.org/.

[14] Borthakur D. The hadoop distributed file system: Architecture and design[J]. 2007.

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

[16]http://hadoop.apache.org/docs/hdfs/current/hdfs_imageviewer.html.

[17]Borthakur, Dhruba. "The hadoop distributed file system: Architecture and design." Hadoop Project Website 11 (2007): 21.

[18] Shvachko, Konstantin, et al. "The hadoop distributed file system." Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010.

[19] DEDEE, FADIKA Z, HARTOG J, etal. MARISSA: MApReduce Implementation for Streaming Science.Applications[C]. E-Science (e-Science), 2012 IEEE 8th International Conference on, 2012:1-8.

[20]郑湃, 崔立真, 王海洋等. 云计算环境下面向数据密集型应用的数据布局策略与方法[J]. 计算机学报, 2010, 08:1472-1480.

[21] http://hadoop.apache.org/docs/r0.19.2/cn/mapred_tutorial.html.

[22] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[23] Thusoo A, Sarma J S, Jain N, et al. Hive-a petabyte scale data warehouse using hadoop[C]//Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010: 996-1005.

[24] https://cwiki.apache.org/confluence/display/Hive/Home.

[25] https://cwiki.apache.org/confluence/display/Hive/LanguageManual.

[26] http://flume.apache.org/FlumeUserGuide.html.

[27] Zawodny J. Redis: Lightweight key/value store that goes the extra mile[J]. Linux Magazine, 2009, 79.

[28] Macedo, Tiago, and Fred Oliveira. Redis Cookbook. O'Reilly Media, 2011.

[29]王利强, 刘正捷, 张丽萍, 等. 网站用户行为数据收集和分析方法[J]. 电脑开发与应用, 2004, 17(2): 56-58.

[30]赵伟, 何丕廉, 陈霞, 等. Web 日志挖掘中的数据预处理技术研究[J]. 计算机应用, 2003, 23(5): 62-67.

[31] 鲍玉斌, 孙焕良, 冷芳玲, 等. 数据仓库环境下以用户为中心的数据清洗过程模型[J]. 计算机科学, 2004, 31(5): 52-55.

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

[33] Gulcu C. Short introduction to log4j[J]. 2002.

[34] Nurseitov N, Paulson M, Reynolds R, et al. Comparison of JSON and XML Data Interchange Formats: A Case Study[J]. CAINE, 2009, 2009: 157-162.

[35] 金健, 陈群, 赵保学. 数据倾斜情况下基于 MapReduce 模型的连接算法研究[J]. 计算机与现代化, 2013 (5): 22-27.

[36] Finnigan P. Using Oracle VPD in The Real World[J]. UKOUG Unix SIG, 2008.

致谢

时光荏苒，两年多的研究生生活即将结束，学生时代的生活也将过去。还记得两年前收到录取通知书的那一刻激动的心情，自己终于如愿进入了心仪的高校进行深造。两年来，自己收获很多，从一个技术小白，到现在略有所成，从对人生道路的懵懵懂懂，到现在的目标明确。

首先我要感谢我的导师王洪波副教授，对学术的严禁认真，值得我们每一个人学习，每次的项目例会，每周的工作总结，王洪波老师都会参加，并指导我们工作的目标和方向。在生活上王洪波老师也对我非常关心，对我的人生观和价值观都产生了积极的影响，再次对老师表示由衷的感谢。

同时感谢我的指导老师邹仕洪副教授，两年多来对我研究生科研的指导和帮助。您教会我如何去进行科研，如何去发现问题，解决问题，在项目中锻炼我们的学习能力和表达能力。您严谨的科研态度和沉稳了研究理念是我学习的榜样，感谢您研究生期间对我的培养。

感谢大师兄刘苑琦，你乐于助人，对我们提出的问题总是不厌其烦的讲解。不仅在学习中给我帮助，在生活上也经常给于指导。感谢已经毕业的师兄师姐在学校给于的各方面帮助和支持。感谢同门，大家一起努力，一起成长。感谢师弟、师妹给予的帮助。感谢学校提供优秀的资源。

最后，感谢我的家人，谢谢你们两年多对我的支持和鼓励。