浙江大学

硕 士 学 位 论 文

论文题目 数据迁移云服务的设计与实现

作者姓名 朱清华

指导教师 陈刚

学科(专业) 计算机应用技术

所在学院 计算机科学与技术

提交日期 2016 年 12 月 27 日

A Dissertation Submitted to Zhejiang University for the Degree of Master of Engineering

TITLE: The Design and Implementation Of Data Migration Cloud Service

Author: Qinghua Zhu

Supervisor: Gang Chen

Subject: Computer Application Technology

College: Computer Science and Technology

Submitted Date: 2016-12-27

摘要

大数据时代的到来，传统的数据存储和处理手段已经难以满足日益增长的需求，越来越多的数据需要迁移到 hadoop 计算平台进行存储和处理。数据迁移作为数据科学领域的重要研究方向和技术，也受到学术界、工业界更多研究人员的关注、研究。

已有的数据迁移工具往往具有着单机性能低下、安装配置繁琐、不支持流式数据迁移等缺点。本文针对现有工具的缺点，结合已有研究成果，设计出了针对 hadoop 集群的数据迁移云服务。本文主要贡献如下：

（1）设计并优化了基于数据库日志的流式数据提取、迁移技术。通过对数据库日志进行解析，提取增量数据，并将这些数据直接封装为消息发往 hadoop 集群。大大降低流式数据提取的 I0、网络等开销。

（2）将因子分析数学思想应用于负载均衡负载状态评估，将响应时间纳入负载均衡参数指标。该算法相对于传统的负载均衡算法，能够更有效地评估节点当前负载情况，更大地利用好集群资源。大大提高了数据迁移系统的吞吐量和集群计算能力。

（3）将数据迁移系统上升到云计算的高度。针对业内已有迁移工具配置复杂、单机性能低下、容错性差等问题，本文提出的数据迁移云服务设计更够更好的提升系统整体迁移能力和吞吐。同时对于迁移任务具有一定的故障可恢复性。

关键词：数据迁移，数据库日志，负载均衡，云计算，分布式

Abstract

With the arrival of era of big data, the traditional solution of data storage and processing has been unable to meet the growing demand. And more and more data need to be migrated to the Hadoop platform for storage and processing. Data migration as an import research area in data science and technology, also concerned by more researchers in academia and industry.

Existing data migration tools have characteristics such as pool performance based on single machine, complicated installation and program failures sometimes. In this paper,the shortcomings of existing tools, combined with the research has successfully Designed a Hadoop cluster's data migration services. Contribution in this paper are as follows.

1)Design optimization based on database and log stream data extraction and migration. By parsing the database logging, incremental data extraction, and package t hedata directly into a message destined for a Hadoop cluster. Greatly reduce the IO stream data extraction and network overhead.

2) Factor analysis and response time are used to assess machines' load condition. Greatly improves the throughput of data migration system and cluster computing pow er.

3) data migration cloud service design is able to better improve the overall transport system capacity and throughput. While migration task has a certain failure recoverability.

Keywords: data migration, database log, load balancing, cloud computing, distributed

目录

图目录

表目录

第1章 绪论

1.1 课题背景

近年来，随着大数据技术、移动互联网、物联网、云计算等新兴技术的发展，大量的数据从终端、移动端、网络端无时无刻不在产生数据。每天产生的数据量惊人，我们从互联网时代迈入数据为主的时代。大数据时代已经到来，很多人已经身处其中，最典型的感觉是数据增加速度之快。数据产生方式现在已经被极大地改变，因为以前数据的生产都是由专业团体、专业人士，或者是专业公司完成，而现在数据产生更多是个体行为、是个人，每个人都可以使用自己所采集的终端来产生大量的数据。

数据传播途径也发生了很大的变革，以前获取信息的来源基本上是报纸等平面媒体，或者电视、广播等传播媒体；现在很多信息来源通过互联网。互联网已经变成了媒体传播的主要途径，这个改变对整个社会也产生了非常大的改变。

社交环境网络化变革，以前交朋友更多是生活的圈子，比如说同学、邻居、亲戚，现在更多的通过是互联网这种虚拟的环境。

数据存储习惯发生变化，以前都是把照片和文件备份到自己的电脑或者软盘上。现在这种观念已经改变，除非做保密工作，或者是年纪大一点的另当别论，大多数人就把它放到网上，在云中进行存储。

在当今企业中 80%的数据都是非结构化数据[1]，这些数据每年都按指数增长 60%。大数据将挑战企业的存储架构、数据中心的基础设施等，也会引发数据仓库、数据挖掘、商业智能、云计算等应用的连锁反应。未来企业会将更多的 TB 级（1TB=1024GB）数据集用于商务智能和商务分析。到 2020 年，全球数据使用量预计将暴增 44 倍，达到 35.2ZB（1ZB=10 亿 TB）。大数据正在彻底改变 IT 世界。

企业沉淀了大量的用户数据，如用户行为数据、用户文件数据，用户数据分析数据报表、日志文件等。而这些数据之前还主要存储在关系型数据库，如 mysql、oracle，或是一些文本文件中。但是已有的关系型数据库、文件已不能满足大数据时代的处理和分析需求，单个节点的性能也非常有限。企业迫切需要将原有的数据迁移到新的分布式平台进行高性能处理，如 hadoop 生态系统。Hive

可以实现大表的分布式 SQL 计算、HDFS 实现大文件可以容错的分布式存储、HBASE 可以实现分布式数据快速索引查询。因此，大数据的飞速发展催生着企业对于数据迁移工具的需求，这个数据迁移的源头是原有的关系型数据，目的地是 hadoop 生态系统，如图：

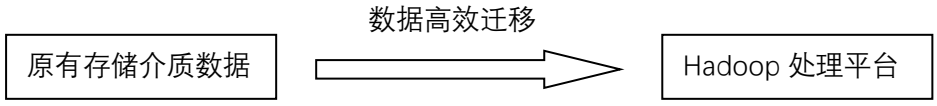

图 1.1 数据迁移示意图

数据迁移是一企业极其看重的一件事，同时又是一件技术含量要求较高的一件事。因为，数据是无价的资产，有关数据的存储和迁移往往对于数据的安全性、稳定性等有着很高的要求。企业对于数据迁移有着如下要求：

- 大量的原有存储介质数据需要迁移到 hadoop 处理平台。这些原有存储介质数据包括了关系型数据库大量业务数据，文件类型的非结构化数据。
- 数据迁移必须确保数据的完整性，传输过程数据不能丢失，数据能够安全完整地迁移到 Hadoop 处理平台。
- 数据迁移尽可能降低对现有的在线业务的影响。比如现有关系数据库也在高速读写，以服务用户请求。数据迁移不能使得原有数据库性能大大下降甚至宕机。
- 数据迁移必须要实现高性能、高可用性。高性能是企业当下业务实际需求，数据的产生和传输都必须以极快的速度。高可用性要求数据迁移业务的不中断性。
- 企业希望能够降低数据迁移的代价，降低数据迁移的学习成本。一个企业往往有若干部门都有着数据迁移的需求。而现有的工具往往配置学习繁琐，若要使用，还得派工程师学习和维护，这大大加剧了不便和问题发生。企业迫切需要简单易配置的数据迁移平台。

云计算技术的发展，为数据迁移性能提高插上了翅膀。2006 年谷歌“Google 101 计划”，并正式提出“云”的概念和理论。随后亚马逊、微软、惠普、雅虎、英特尔、IBM 等公司都宣布了自己的“云计划”，云安全、云存储、内部云、外

部云、公共云、私有云。云计算(Cloud Computing)是由分布式计算(Distributed Computing)、并行处理 (Parallel Computing)、网格计算 (Grid Computing) 发展来的，是一种新兴的商业计算模型。

云计算技术，利用集群化整合的存储和计算能力，能够将复杂需要大量计算任务得到更快的处理。同时，云计算提供的是一个平台，为用户屏蔽底层实现，这位用户的使用提供了大大便捷。

将云计算用于数据迁移服务中，可以大大提高企业内部众多数据迁移任务的性能。同时基于软件即服务的思想，云计算提供出的数据迁移平台将会给企业业务人员使用带来大大便捷。同时作为一种公有平台，也更容易集中精力去维护。

1.2 研究现状

目前工业界和学术界对于数据迁移问题，即将数据从关系型数据库、日志文件迁移到 hadoop 处理平台研究得不胜热烈。目前的数据迁移主要集中在静态数据迁移和动态数据迁移，也可分为全量数据迁移、增量数据迁移、流式数据迁移，还可以包括离线数据迁移和在线数据迁移。

对于静态数据迁移，研究得相对成熟。所谓静态数据迁移，是指数据源头不发生变化，且往往是以全量的方式将数据源指定数据完整地迁移到 hadoop 平台。这里由于数据源的静态性使得数据迁移系统可以一次性获取数据源全部数据，而不需要实时监听，所以性能还不错。缺点时这种迁移局限性大，因为实际业务数据库数据源数据往往在不断变化。

动态数据迁移，指数据源可能在不断变化，或是迁移并不是一次完成。这对技术的要求就更高了，需要实时监听数据源的变化，抽取增量数据，实现传输。而这正符合企业实际业务场景。

在线数据迁移，是指将正在提供线上服务的数据，从一个地方迁移到另一个地方，整个迁移过程中要求不停机，服务不受影响。根据数据所处层次，可以分为 cache 迁移和存储迁移；根据数据迁移前后的变化，又可以分为平移和转移。在线数据迁移最大的挑战是如何保证迁移过程服务不受影响。

迁移策略有一下几种：

- 一次迁移[2]是通过数据迁移工具或迁移程序，将需要的历史数据一次性全部迁移到新系统中。一次迁移的优点是迁移实施的过程短，相对分次

迁移，迁移时涉及的问题少，风险相对比较低。其缺点是工作强度比较大，由于实施迁移的人员需要一直监控迁移的过程，如果迁移所需的时间比较长，工作人员会很疲劳。一次迁移的前提是新旧系统数据库差异不大，在允许的宕机时间内可以完成所有数据量的迁移。

- 分次迁移[3]是通过数据迁移工具或迁移程序，将需要的历史数据分几次迁移到新系统中。分次迁移可以将任务分开，有效地解决数据量大和宕机时间短之间的矛盾。但是分次切换导致数据多次合并，增加了出错的概率，同时为了保持整体数据的一致性，分次迁移时需要对先迁移的数据进行同步，增加了迁移的复杂度。分次迁移一般在系统切换前先迁移静态数据和变化不频繁的数据，例如代码、用户信息等；然后在系统切换时迁移动态数据，例如交易信息。对于静态数据迁移之后发生的数据变更，可以每天同步到新系统中，也可以在系统切换时通过增量的方式一次同步到新系统中。
- 先录后迁是在系统切换前，先通过手工把一些数据录入到新系统中，系统切换时再迁移其他的历史数据。先录后迁主要针对新旧系统数据结构存在特定差异的情况，即对于新系统启用时必需的初始数据，无法从现有的历史数据中得到。对于这部分初始数据，就可以在系统切换前通过手工录入。
- 先迁后补是指在系统切换前通过数据迁移工具或迁移程序，将原始数据迁移到新系统中，然后通过新系统的相关功能，或为此专门编写的配套程序，根据已经迁移到新系统中的原始数据，生成所需要的结果数据。先迁后补可以减少迁移的数据量。

Hadoop 是由 Apache 基金会所开发的分布式处理平台。其最核心的设计就是：HDFS 分布式文件系统和分布式计算框架 Map/Reduce。HDFS 具有高容错性、高吞吐量的特点，为海量数据提供存储；Map/Reduce 则为大数据数据计算提供支持。目前，Hadoop 以其低成本、高扩展性、靠可靠性以及靠容错性等优点，成为新一代的大数据处理平台。很多公司也开始提供基于 Hadoop 的商业软件、支持、服务以及培训。据估计，每年 Hadoop 的销售额会增长近 60%，到 2020 年会达到

500 亿左右。随着越来越多的公司开始使用 Hadoop 产品，大量的数据迁移工作由此产生。

理论上讲，Hadoop 平台数据的迁移（包括迁入和迁出）是相关软件以及用户就可以完成的工作。例如，Apache 的 Sqoop[4]就是一个用来将 Hadoop 和关系型数据库中的数据相互转移的工具。它可以将一个关系型数据库（例如 ：MySQL ,Oracle ,Postgres 等)中的数据导入到 Hadoop 的 HDFS 中，也可以将 HDFS 的数据导入到关系型数据库中。然而，大数据时代是的数据迁移需要耗费大量的人力。于是，Hadoop 相关的数据迁移工具和服务的需求相应增加。

为了能够方便数据迁移，无论是原有的数据迁移或者数据备份工具相关的公司，还是 Hadoop 厂商都开始提供相应的产品和服务。例如，Attunity 公司推出的数据复制产品——Attunity Replicate[5]。该产品针对 EMC Greenplum 多款产品进行了性能优化。除了 Hadoop，它所支持的数据复制/迁移平台还包括 Oracle、DB2、SQL Server、Greenplum 以及 Teradata 等。此外，Diyotta DataMover 也同样支持 Hadoop 平台中多种格式的数据迁入和迁出。大型机数据集成厂商 Syncsort 已经与 Hadoop 厂商 Cloudera 宣布合作，准备将大型机数据与 Hadoop 集群更紧密地联系起来，从而进行大数据分析。据 Syncsort 总裁 Josh Rogers 预测，将大型机负载逐渐迁移到 Hadoop 集群将是未来 Hadoop 在企业中的一个主要应用场景。

可以看出，基本上所有的数据迁移工具和服务都能够支持多数平台间的数据迁移。那么，方便性和整体服务就成为了提高产品竞争力的重要方面。像 Hortonworks 这样的 Hadoop 厂商充分利用自身的优势，已经开始推出自己的迁移支持和服务。这样，Hadoop 厂商就可以在数据迁移工具和服务方面占据自己的市场，避免让 Sqoop 这样的产品成为 Hadoop 平台数据迁移中必须的第三方工具。

在 Hadoop 相关的数据迁移工具和服务激烈的竞争中，寻求更好的设计理念，并能够把产品设计的目光放得更加长远就十分关键。在现在的设计中，能够为未来可能的变化预留接口。例如，提供对未来的 Hadoop 数据安全框架 Apache Argus 的支持就是十分重要的一方面。总的来讲，最好的 Hadoop 数据迁移方面的长期投资还在于理解目前已经存在的工具，然后结合其中的优点创造出更能满足用户

需求产品。

总结现有数据迁移工具缺点如下：

- 数据迁移性能有限。现有工具往往只运行在一个计算节点，单机运算，CPU 运算能力和网络传输能力很有限。使得数据迁移性能不高。
- 迁移过程具有盲目性。程序一旦崩溃，往往没有合适的故障恢复机制。数据迁移任务一旦中断，数据传输过程就需要重头再来。大大耗费了计算资源和人力维护成本。
- 数据迁移缺少统一的管理平台。现有的传输工具往往零散，各自为战，都在宣传自己的优势。用户学习成本大，维护成本高。软件需要独立安装。
- 数据迁移缺少调度模块。实际业务流之间往往有先后依赖关系，而现有工具只着眼于传输，缺乏全局统一调度。使得数据传输作业混乱。混乱的结果就是整个系统吞吐量不高。
- 对于流式数据迁移支持的不够好，缺乏较好的方式。对于全量数据迁移已有了不错的效果，而实际企业业务场景，数据源在实时变化。数据流式活的而不是死的，数据应该以一种流式方式实时地流向 hadoop 生态系统。

1.3 本文主要工作

本文针对现有业内流行的数据迁移工具的优缺点，做了大量的调研和学习工作。深入了解了现有工具的优缺点。并从工业界的实际需求出发，设计数据迁移云服务系统。即把关系型数据库结构化数据、文件非结构化数据迁移到 Hadoop 数据处理平台，以供下一步处理和分析。论题具有较大的实际应用价值。

论文结合了云计算、大数据、分布式理论、监控等理论，以期能够设计出安全、高效、故障自恢复、容错、高可用的数据迁移平台。且大大简化配置，简单易用。

本文研究工作涉及数据迁移、云计算、负载均衡算法等多方面，主要研究工作有：

- 本文所设计的数据迁移云服务紧贴企业实际需求，包含了数据迁移最核心的功能。如全量数据迁移，即一次性把数据源指定数据全部迁移到 hadoop 生

态系统；增量数据迁移，即增量地把数据源指定数据分批地迁入到 hadoop 处理平台；流式数据迁移，这也是论文所论述的重点。流式数据迁移，也是类似实时数据迁移，即数据源在实时变化，系统能够自动收集增量数据并进行周期性迁移到 hadoop 平台。整个数据以水流形式高效进入 hadoop 生态系统。

- 针对数据库流式数据传输的难点，本文创新性地提出了基于数据库日志文件增量数据提取的流式数据迁移方案。增量数据提取是流式数据迁移的关键所在。即怎样把新数据高效抽取出来。本文基于现有企业数据库模式和特点，通过解析数据库日志文件，高效抽取新增数据，封装数据发往 hadoop 平台。本方法使得流式数据提取有效，而且对于数据源的影响非常低，数据源业务正常开展。做到了在线迁移、流式迁移。
- 针对现有数据迁移工具缺乏有力的任务管理平台。本文所论述的数据迁移云服务，包含了功能强大、简单易用的任务管理平台。企业数据迁移任务众多，我们希望任何迁移任务都要能够查看，如任务执行状态、任务完成时间、任务已完成百分比、故障异常信息查看等功能。真正做到，对于数据迁移的全方位细致管理。使得对传统数据迁移工具不懂的业务人员也能尽快上手该服务。
- 提出了以拓扑排序算法为主的迁移作业流管理方法。实际的数据迁移完整流程除了数据传输的关键阶段，可能还包括了数据的简单过滤预处理，分阶段性；同时，任务与任务之间很可能有着依赖关系，比如希望有的任务先执行，有的数据之后导入。本文所设计的作业流管理，使得用户只需要输入任务与任务之间的依赖关系，系统即会根据输入数据做拓扑排序。得到最优化的作业执顺序，把能够并行执行的作业并行执行以提高系统性能，有依赖关系的作业则顺序执行。最大化程度既满足用户的实际需求，又提高了系统整体的运行效率。
- 本文设计了强大的系统监控和故障自恢复系统。针对分布式集群运行环境的复杂性和不确定性，本文所设计的监控系统能够最大程度减少数据迁移服务的中断。当某个服务节点出现了异常或宕机，监控系统能够立即发现，并进行服务切换使得迁移任务继续进行。对于系统关键性执行性进程，通过心跳

等措施周期性观察。同时，本文还设计了故障自恢复机制。即当数据迁移任务中断时，因为有日志文件记录当前任务的执行状态和进度，待进程重启或是节点切换，数据迁移仍能够继续进行。

- 为了提高系统整体任务吞吐量。本文提出了基于因子分析和响应时间的负载均衡算法。该算法相对于传统的负载均衡算法，能够更好地评估当前节点负载状态，并且在任务实际执行过程中动态调整节点权重，防止数据倾斜、任务倾斜。更科学地调度任务，发挥集群计算能力。
- 本文将传统的数据迁移，单机化操作，上升到云计算服务层面。即数据迁移作为一种云服务提供给用户。云计算将会利用集群整体的计算能力，把数据迁移任务执行、监控管理等模块的性能提高到一个很高的高度。大大提高了性能。同时大大简化了用户的学习成本。用户通过 web 界面，进行任务申请、配置填写即可。中间迁移过程用户可全程查看。方便易用。

1.4 论文结构

本文有七章，其组织结构如下：

- 第一章，介绍了数据迁移研究课题的背景及意义。并介绍了国内外学术界、工业界研究现状，列出了现有数据迁移工具的一些缺点。接下来介绍了本文的主要工作和贡献。
- 第二章，介绍了论文设计方案的相关技术。从基本的数据迁移技术，到工业界已有的数据传输工具特性原理分析，关系型数据库增量数据提取技术等。这章为后来几章技术设计做了铺垫和叙述。
- 第三章，从数据迁移的实际需求出发，做了顶层架构设计。描述了系统的功能模块划分，包括数据迁移模块、任务管理和调度模块、监控报警模块。
- 第四章，详细介绍数据迁移云服务各模块的设计与实现。包括了设计过程中的问题和挑战，实现包括了代码、逻辑、原理。
- 第五章，针对前文所叙述，设计了具有针对性的实验。并对实验结果进行了分析，以反映文中设计的特点。
- 第六章，对本文进行生化总结。对未来工作进行展望。

1.5 本章小结

本章从数据迁移的研究背景出发，剖析了课题研究意义，接着分析了国内外研究成果，指出了工业界已有解决方案的不足。概括了本文所论述系统的特性，最后对论文工作和结构做了介绍。

第2章 相关理论与技术

2.1 数据迁移技术

典型的 Web 应用程序是采用三层架构构建的[6]。应用程序要向外扩展时，一般是简单地在负载均衡器之后添加更多的商品化 Web 服务器来支持更多用户。而对于越来越重要的云计算模型而言，向外扩展能力是其核心原则。在云计算模型中，虚拟机实例很容易根据需求进行相应地添加或删除。

然而，当涉及到数据层时，关系数据库（RDBMS）不但无法向外扩展，也没有提供灵活的数据模型，这方面有很多挑战。要处理更多的用户，这意味着要加入一台更为大型的服务器，而大型的服务器复杂度很高，一般是专有的而且非常昂贵，不像基于 Web 或云的架构中所使用的商品化硬件那样廉价。因此，当公司开始发现现有应用或新应用所用的关系数据库存在性能问题时，特别是这一切与用户数目的增长有关时，他们意识到需要一个更快的、有弹性的数据库层。现在是时候评估 NoSQL 技术并将其作为交互式 Web 应用的数据库层了。事实上，大多数情况下并不是放弃 SQL 转而寻求 NoSQL[7]解决方案，而是为了让应用和用例满足需求的变化，从一种方案转向另一种方案。一般而言，在构建现代 Web 和移动应用时，不管是伸缩模型还是数据模型，对灵活性都有特定的需求，而这种需求正是从 SQL 向 NoSQL 迁移的推动因素。

近年来，由于 hadoop 开源软件[8]的兴起，企业越来越多的数据存储和处理工作放到了 hadoop 生态系统。而已有的大量日志文件、业务数据还存储在文件、关系型数据库等传统存储介质上。因此，迫切需要迁移工具能够将原有数据安全、高效地迁移至 hadoop 处理平台。

数据迁移系统的设计需要考虑性能问题。数据迁移的量往往很大，而且不能对在线业务数据运转造成太大影响。数据迁移往往在合适的时段进行，如晚上用户请求不那么频繁的时间段。其次应当考虑，数据迁移的成本问题。比如工具的学习成本、人力成本、维护成本，如果成本过高，就体现不出 Hadoop 廉价服务的优势。还要考虑风险因素，数据的安全性和完整性。数据是无价的资产，数据迁移过程能否有效保护数据源不受损害，传输过程数据不遗失。

2.2 关系型数据库日志

关系型数据库数据迁移至 hadoop 平台是数据迁移的重点。因此有必要研究关系型数据库数据增量提取技术。而记录数据库变化的，往往通过数据库日志文件。

对于 mysql 数据库，常常通过查看 binlog[9]文件来查看数据库数据变动。binlog 是二进制日志文件，用于记录 mysql 的数据更新或者潜在更新(比如 DELETE 语句执行删除而实际并没有符合条件的数据)，在 mysql 主从复制中就是依靠的 binlog。在 mysql 中开启 binlog 需要设置 my.cnf 中的 log_bin 参数，另外也可以通过 binlog_do_db 指定要记录 binlog 的数据库和 binlog_ignore_db 指定不记录 binlog 的数据库。对运行中的 mysql 要启用 binlog 可以通过命令 SET SQL_LOG_BIN=1 来设置。设置完成，我们就可以来测试 binlog 了。显然，我们执行 SELECT 等不涉及数据更新的语句是不会记 binlog 的，而涉及到数据更新则会记录。要注意的是，对支持事务的引擎如 innodb 而言，必须要提交了事务才会记录 binlog。

binlog 刷新到磁盘的时机跟 sync_binlog 参数相关，如果设置为 0，则表示 MySQL 不控制 binlog 的刷新，由文件系统去控制它缓存的刷新，而如果设置为不为 0 的值则表示每 sync_binlog 次事务，MySQL 调用文件系统的刷新操作刷新 binlog 到磁盘中。设为 1 是最安全的，在系统故障时最多丢失一个事务的更新，但是会对性能有所影响，一般情况下会设置为 100 或者 0，牺牲一定的一致性来获取更好的性能。

binlog 格式分为 statement，row 以及 mixed 三种[10]，mysql5.5 默认的还是 statement 模式，当然我们在主从同步中一般是不建议用 statement 模式的，因为会有些语句不支持，比如语句中包含 UUID 函数，以及 LOAD DATA IN FILE 语句等，一般推荐的是 mixed 格式。暂且不管这三种格式的区别，看看 binlog 的存储格式是什么样的。binlog 是一个二进制文件集合，当然除了我们看到的 mysql-bin.xxxxxx 这些 binlog 文件外，还有个 binlog 索引文件 mysql-bin.index。如官方文档中所写，binlog 格式如下：

- binlog 文件以一个值为 0Xfe62696e 的魔数开头，这个魔数对应 0xfe 'b''i''n'。

- binlog 由一系列的 binlog event 构成。每个 binlog event 包含 header 和 data 两部分。header 部分提供的是 event 的公共的类型信息，包括 event 的创建时间，服务器等等。data 部分提供的是针对该 event 的具体信息，如具体数据的修改。
- 从 mysql5.0 版本开始，binlog 采用的是 v4 版本，第一个 event 都是 format_desc event 用于描述 binlog 文件的格式版本，这个格式就是 event 写入 binlog 文件的格式。
- 接下来的 event 就是按照上面的格式版本写入的 event。最后一个 rotate event 用于说明下一个 binlog 文件。binlog 索引文件是一个文本文件，其中内容为当前的 binlog 文件列表。

Event 数据结构如下：

表 2.1 Event 数据结构

Event header	Timestamp
	Type_code
	Server_id
	Event_length
	Next_position
	Flags
	Extra_headers
Event data	Fixed part
	Variable part

将 MySQL 日志格式设置为 row 级时，在 test 数据库下创建一个表格。如”create table students(id int,name varchar)”;插入语句”insert into students values(2,a)”。这将会产生一个 query event,一个 table_map event、一个 write_rows event 以及一个 xid event。内容如下：

表 2.2 Event 详细内容

Pos	Event_type	Server_id	End_log_pos	Info
4	Format_desc	4	107	Binlog ver: 4
107	Query	4	175	BEGIN
175	Table_map	4	221	Table_id:50 (test.students)
221	Write_rows	4	262	Table_id:50 Flags:STMT_END
262	Xid	4	289	Commit /* xid=245*/

对应的 binlog 二进制文件如下:

```
#query event (BEGIN)
0000006b    95 2a 85 56 02 04 00 00    00 44 00 00 00 af 00 00    |.*.V.....D......|
0000007b    00 08 00 26 00 00 00 00    00 00 00 04 00 00 1a 00    |...&............|
0000008b    00 00 00 00 00 01 00 00    00 00 00 00 00 00 06 03    |................|
0000009b    73 74 64 04 21 00 21 00    08 00 74 65 73 74 00 42    |std.!.!...test.B|
000000ab    45 47 49 4e                                           |EGIN|

#table_map event
000000af    95 2a 85 56 13 04 00 00    00 2e 00 00 00 dd 00 00    |.*.V............|
000000bf    00 00 00 32 00 00 00 00    00 01 00 04 74 65 73 74    |...2........test|
000000cf    00 04 74 72 6f 77 00 02    03 0f 02 0a 00 02          |..students......|

#write_rows event
000000dd    95 2a 85 56 17 04 00 00    00 29 00 00 00 06 01 00    |.*.V.....)......|
```

0x0000000af 开始为 table_map event。除去头部 19 个字节，Fixed data 为 8 个字节，前面 6 个字节 0x32=50 为 table id，接着 2 个字节 0x0001 为 flags。Variable data 部分，首先 1 个字节 0x04 为数据库名 test 的长度，然后 5 个字节是数据库名 test+结束符。接着 1 个字节 0x04 为表名长度，接着 5 个字节为表名 students+结束符。接着 1 个字节 0x02 为列的数目。而后是 2 个列的类型定义，分别是 0x03 和 0x0f（列的类型 MYSQL_TYPE_LONG 为 0x03，MYSQL_TYPE_VARCHAR 为 0x0f)。接着是列的元数据定义，首先 0x02 表示元数据长度为 2，因为 MYSQL_TYPE_LONG 没有元数据，而 MYSQL_TYPE_VARCHAR 元数据长度为 2。接着的 0x000a 就是 MYSQL_TYPE_VARCHAR 的元数据，表示我们在定义表时的 varchar 字段 c 长度为 10，最后一个字节 0x02 为掩码，表示第一个字段 i 不能为 NULL。从 0x000000dd 开始为 write_rows event，除去头部 19 个字节，前 6 个字节 0x32 也是 table id，然后两个字节 0x0001 为 flags。接着的 1 个字节 0x02 为表中列的数目。然后 1 个字节 0xff 各个 bit 标识各列是否存在值，这里表示都存在。

接着的就是插入的各行数据了。第 1 个字节 0xfe 的各个 bit 标识该行变化之后各列是否为 NULL，为 NULL 记为 1.这里表示第 1 列不为 NULL，因为第一行数据插入的是(1,NULL)。接下来是各列的数据，第一列是 MYSQL_TYPE_LONG,长度为 4 个字节，所以 0x00000001 就是这个值。第二列是 NULL 不占字节。接下来是第二行,先是 0xfc 标识两列都不为 NULL,先读取第一列的 4 个字节 0x00000002 也就是我们插入的数字 2，然后读取第二列，先是一个字节的长度 0x01，然后是内容 0x61 也就是字符'a'。

对于 binlog 的三种模式，各有优缺点，如下:

- statement：基于 SQL 语句的模式，某些语句和函数如 UUID, LOAD DATA INFILE 等在复制过程可能导致数据不一致甚至出错。
- row：基于行的模式，记录的是行的变化，很安全。但是 binlog 会比其他两种模式大很多，在一些大表中清除大量数据时在 binlog 中会生成很多条语句，可能导致从库延迟变大。
- mixed：混合模式，根据语句来选用是 statement 还是 row 模式。

对于论文中所采用的方案是 row 级日志记录模式。因为此种模式能够把所有

数据变更详细记录在内，不会造成增量数据遗失情况。

2.3 负载均衡技术

负载均衡[11]，顾名思义，主要使用在业务量和数据量较高的情况下，当单台服务器不足以承担所有的负载压力时，通过负载均衡手段，将流量和数据分摊到一个集群组成的多台服务器上，以提高整体的负载处理能力。常见的负载均衡算法如下：

Random 随机[12]，按权重设置随机概率。在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

轮询(Round Robbin)[13]当服务器群中各服务器的处理能力相同时，且每笔业务处理量差异不大时，最适合使用这种算法。 轮循，按公约后的权重设置轮循比率。存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。加权轮询(Weighted Round Robbin)为轮询中的每台服务器附加一定权重的算法。

最少连接(Least Connections)[14]在多个服务器中，与处理连接数(会话数)最少的服务器进行通信的算法。即使在每台服务器处理能力各不相同，每笔业务处理量也不相同的情况下，也能够在一定程度上降低服务器的负载。

加权最少连接(Weighted Least Connection)[15]为最少连接算法中的每台服务器附加权重的算法，该算法事先为每台服务器分配处理连接的数量，并将客户端请求转至连接数最少的服务器上。

2.4 Zookeeper

ZooKeeper[16]是一个分布式的，开放源码的分布式应用程序协调服务，它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务，配置维护和命名服务等。在分布式应用中，由于工程师不能很好地使用锁机制，以及基于消息的协调机制不适合在某些应用中使用，因此需要有一种可靠的、可扩展的、分布式的、可配置的协调机制来统一系统的状态。Zookeeper 的目的就在于此。

Zookeeper 的核心是原子广播，这个机制保证了各个 Server 之间的同步。实现这个机制的协议叫做 Zab 协议。Zab 协议有两种模式，它们分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab 就进入了恢复模式，当领导者被选举出来，且大多数 Server 完成了和 leader 的状态同步

以后，恢复模式就结束了。状态同步保证了 leader 和 Server 具有相同的系统状态。

为了保证事务的顺序一致性，zookeeper 采用了递增的事务 id 号（zxid）来标识事务。所有的提议（proposal）都在被提出的时候加上了 zxid。实现中 zxid 是一个 64 位的数字，它高 32 位是 epoch 用来标识 leader 关系是否改变，每次一个 leader 被选出来，它都会有一个新的 epoch，标识当前属于那个 leader 的统治时期。低 32 位用于递增计数。

每个 Server 在工作过程中有三种状态：

- LOOKING：当前 Server 不知道 leader 是谁，正在搜寻。
- LEADING：当前 Server 即为选举出来的 leader。
- FOLLOWING：leader 已经选举出来，当前 Server 与之同步。

Zookeeper 典型应用场景如下:

- 数据发布与订阅。发布与订阅即所谓的配置管理，顾名思义就是将数据发布到 zk 节点上，供订阅者动态获取数据，实现配置信息的集中式管理和动态更新。例如全局的配置信息，地址列表等就非常适合使用。
- 分布式通知与协调。ZooKeeper 中特有 watcher 注册与异步通知机制，能够很好的实现分布式环境下不同系统之间的通知与协调，实现对数据变更的实时处理。
- 分布式锁。通常的做法是把 zk 上的一个 znode 看作是一把锁，通过 create znode 的方式来实现。所有客户端都去创建 /distribute_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。
- 集群管理。如集群监控和 master 节点选举。

2.5 云计算

云计算[17]是基于互联网的相关服务的增加、使用和交付模式，通常涉及通过互联网来提供动态易扩展且经常是虚拟化的资源。云是网络、互联网的一种比喻说法。过去在图中往往用云来表示电信网，后来也用来表示互联网和底层基础设施的抽象。因此，云计算甚至可以让你体验每秒 10 万亿次的运算能力，拥有这么强大的计算能力可以模拟核爆炸、预测气候变化和市场发展趋势。用户通过电脑、笔记本、手机等方式接入数据中心，按自己的需求进行运算。

对云计算的定义有多种说法。对于到底什么是云计算，至少可以找到 100 种解释。目前广为接受的是美国国家标准与技术研究院（NIST）定义：云计算是一种按使用量付费的模式，这种模式提供可用的、便捷的、按需的网络访问，进入可配置的计算资源共享池（资源包括网络，服务器，存储，应用软件，服务），这些资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互。

云计算可以认为包括以下几个层次的服务：基础设施即服务（IaaS），平台即服务（PaaS）和软件即服务（SaaS）。IaaS：基础设施即服务 IaaS(Infrastructure-as-a- Service)。消费者通过 Internet 可以从完善的计算机基础设施获得服务。PaaS：平台即服务 PaaS(Platform-as-a- Service)：。PaaS 实际上是指将软件研发的平台作为一种服务，以 SaaS 的模式提交给用户。因此，PaaS 也是 SaaS 模式的一种应用。但是，PaaS 的出现可以加快 SaaS 的发展，尤其是加快 SaaS 应用的开发速度。SaaS(Software-as-a- Service)：软件即服务。它是一种通过 Internet 提供软件的模式，用户无需购买软件，而是向提供商租用基于 Web 的软件，来管理企业经营活动。

2.6 本章小结

本章主要介绍了数据迁移云服务设计中设计到的重要理论和技术。包括了数据迁移技术、mysql 数据库日志文件格式解析、常见的负载均衡算法、以及分布式云计算技术。这些基础理论技术，正式为了设计出易用强大的数据迁移云服务。

第3章 数据迁移云服务总体设计

本章主要介绍整个数据迁移云服务的总体设计和组成架构，并对架构各个组件模块的需求功能和设计思想进行论述。

3.1 系统总体需求

在数据量急剧膨胀的今天，企业现有的数据库和数据查询分析系统已不能满足高并发的应用场景。原有结构化数据如关系型数据库数据，和非结构化数据如 CSV，EXCEL 等文件都需要迁移到分布式存储处理平台进行存储和分析。本文所设计的数据迁移云服务，紧贴企业的实际需求，针对现有数据迁移系统在使用上和维护上的不足提出了更完善的架构设计。目的是提高数据迁移系统的高可用性、高可靠性，大大简化数据迁移操作配置，以云服务的形式提供给业务部门。其主要功能需求如下：

1. 数据迁移实现传统存储平台的数据迁移到 hadoop 平台。实现结构化数据如关系型数据库数据 mysql、oracle 等数据，非结构化数据如 csv,excel 等文本文件数据迁移到 hadoop 生态系统如 HDFS,HIVE,HBASE。关于数据迁移的形式，应包括全量数据迁移，即把指定数据源数据全部迁移到 hadoop 分布式存储平台；增量数据迁移，即把指定数据源根据用户指定的标准仅把增量数据迁移到 hadoop 分布式存储平台；流式数据迁移，即把指定数据源数据流式地、实时地迁移到 hadoop 分布式存储平台。
2. 设计一个高可用、高可靠、高吞吐的数据迁移云服务系统。数据迁移云服务平台能够运行若干数据迁移任务。一个公司有若干部门有着数据迁移的需求，所以该数据迁移系统以云服务提供给业务使用人员。
3. 合理有效的迁移任务管理和调度系统。业务人员对于迁移任务可通过简单、使用的页面配置，实现后台任务。迁移任务配置包含了主要数据源、数据目的、简单的数据过滤、任务执行模式等。同时要有迁移调度模块。因为实际企业业务有着任务调度的需求，比如数据迁移任务之间可能有依赖关系，有的迁移任务希望先执行，有的迁移任务希望周期执行。
4. 科学有效地监控与故障报警恢复系统。作为一个重要的分布式云服务，设计多个节点计算，数据传输。一个高可用高可靠的云服务一定要有科学有效地

监控报警系统。对于数据迁移的任何环节都应当予以监控，使得迁移过程可控可视。一旦数据迁移服务的某个节点或环节出现问题，应该能够及时反馈给运维人员。及时采取措施进行补救。

数据迁移云服务应该具备的特性：

- 多源异构的支持。在企业信息化建设过程中，由于各业务系统建设和实施数据管理系统的阶段性、技术性以及其它经济和人为因素等因素影响，导致企业在发展过程中积累了大量采用不同存储方式的业务数据，包括采用的数据管理系统也大不相同，从简单的文件数据库到复杂的网络数据库，它们构成了企业的异构数据源。本文论述的数据迁移云服务要能实现对多源异构的支持，如数据库结构化数据 mysql,oracle 等，非结构化数据文本文件、日志等。
- 强大的监控和故障恢复报警系统。本文所设计的数据迁移云服务使得任何一个数据迁移任务是可以追踪的，包括任务的当前执行状态、完成程度百分比。云服务系统设计要充分考虑分布式系统的不确定性，认为任何环节都可能出现死机、宕机等问题。一个数据迁移任务即使死掉了也要能恢复重启继续进行任务，对不可恢复的严重故障要及时通过邮件或短信通知运维和系统人员，及时报警解决。
- 高可用性。一个企业内部众多部门都有着数据迁移的实际业务需求，数据迁移云服务承担着量大、庞杂的数据迁移任务。数据迁移云服务的健壮程度、高可用性是衡量云服务架构的关键指标之一。而实际的业务部门，尤其是流式数据迁移更是对云服务系统的高可用性有着较高的要求。分布式系统的复杂性包括网络环境的复杂性、服务器物理设备健康程度的不确定性、节点的负载和计算能力不稳定性等，这些因素增添了分布式云服务的不确定性。一个完善的云服务必将花很多精力考虑高可用的设计。当数据源捕获服务、数据过滤和解析服务、数据迁移执行服务等节点出现错误甚至宕机时，要能够立即对服务进行重启或切换，确保数据迁移服务的正常进行。
- 高并发能力。数据迁移云服务中的高并发能力一是实际企业业务的需求，因为企业众多部门均有数据迁移的需求，且每个部门每天都有众多数据

迁移的任务。实际的业务场景所需。二是，高并发能力也是云服务高性能的体现之一。多于大量的迁移任务，高并发能力是最大限度利用好整个分布式云服务系统的资源，包括网络带宽、节点的计算和存储能力，最大限度地提高数据迁移的效率和速率，大大缩减业务部门数据迁移的时间代价。对于流式数据迁移，更是如此。

- 高可扩展性。由于企业业务的快速发展，数据量以一种不可估计的速度在膨胀，企业的功能需求也在不断变化。最初的节点计算能力和分布式资源可能并不能满足后期企业的业务需求。因此不太容易一开始就提出一个完美的架构和设计方案。任何一个强大的分布式云服务都是一个不断优化和改进的过程。那么一个成功的云服务是希望能够通过简单的改进来扩展系统的能力，比如通过增加几台执行服务器就能够对数据迁移云服务的性能有所提高。数据迁移云服务除了对物理硬件线性扩展的良好支持，同时对功能模块扩展也要有很好的兼容。随着项目的规模越来越大，项目的维护性就可能会变得越来越差，有时可能会出现牵一发而动全身的情况。如果需要修改某个功能的代码，或者添加某项功能，会耗费大量的人力和时间。这种情况下，高可扩展性的、低耦合的应用程序就变得非常重要了。
- 高吞吐能力[18]。数据迁移云服务的吞吐量指的是单位时间所能接受处理的任务请求数量，更大的吞吐量意味着云服务能够同时处理的数据迁移任务数更多，系统运载能力更强大。一个系统吞吐量通常由 QPS（每秒钟事务处理数量）、并发数两个因素决定，受到硬件物理资源和架构设计好坏有关。本文所论述的云服务架构设计力求把系统吞吐量提高到很大，这样能够更好的满足企业需求。

3.2 系统整体架构设计

整个数据迁移云服务系统有多个模块组成，包括了云服务前台任务接受模块、迁移任务管理和存储模块、负载均衡模块、数据迁移任务执行模块、监控和报警模块等。

整个云服务架构设计从用户实际需求出发，能够满足多用户提交、多任务运行调度管理、任务监控、集群故障报警功能。

前台任务接受模块是用户与整个数据迁移云服务交互的第一块。当用户有着数据迁移的需求时，通过前台 web 界面配置数据迁移任务信息。包括了数据源，数据目的地，数据迁移方式等。前台接受模块收到用户请求后执行相应的负载均衡调度算法，分配最合适的执行服务节点进行数据迁移服务。

任务管理和存储模块负责数据迁移任务信息的持久化和管理。为了数据迁移任务的查询和中断恢复，数据迁移任务配置、执行信息必须保存到物理介质进行持久化。这也是数据迁移任务管理的前提基础。

数据迁移任务执行模块是整个数据迁移服务系统的核心。负责数据迁移的任务执行，如全量数据迁移、增量数据迁移、流式数据迁移。其包括了增量数据提取模块、数据解析与过滤模块、数据传输模块、远程调用模块等。

监控与报警模块也是整个云服务系统重要一部分。整个分布式系统的不确定性决定着一个健壮的云服务系统一定要有监控模块。该模块负责整个系统节点的健康状况检查、任务执行状态跟踪检查、任务中断与恢复、重大故障报警。

整个架构从企业实际需求出发，丰富了数据迁移云服务功能。同时科学的考虑了系统的健壮性和可用性，使得整个系统具备一定的故障恢复能力。模块设计也尽可能的考虑高内聚低耦合，这也为系统的可扩展性提供了基础。

整体架构图如下：

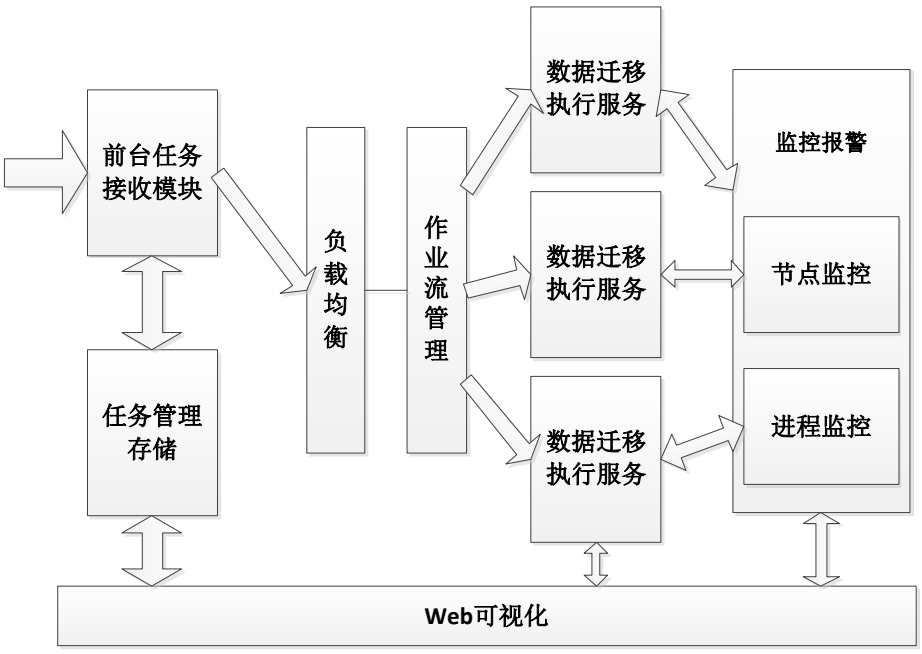

图 3.1 数据迁移整体框架图

3.3 数据迁移模块设计

数据迁移模块是整个云服务的核心模块。根据企业业务实际需求，迁移形式主要有三种。全量数据迁移、增量数据迁移、流式数据迁移。

3.3.1 全量数据迁移

全量数据迁移指的是对于某个数据源数据某个时刻的全部数据迁移到 hadoop 集群，如某个时刻将某个日志文件的所有内容或是某个数据库整张表数据迁移到 hadoop 生态系统。这是一种静态数据迁移，数据源从一开始就是固定的，这种迁移往往是一次性迁移完成，这种迁移也相对容易控制和实现。

3.3.2 增量数据迁移

增量数据传输也是非常有必要的。因为数据源的数据量可能巨大，比如文件众多或是数据库表中表众多且单表很大，可能需要分多次迁移。

增量数据的迁移首先要考虑如何区分老数据和新数据。对于日志文件这样的非结构化数据，可以把读取日志文件的位置作为区分标准。位置之前的数据已经迁移完，作为老数据；位置之后的，还没有迁移，作为新数据。这个位置可以是文件行号，也可以是二进制文件中的特殊位置标记。对数据库结构化数据，通常可以以表中某个字段或是主键为依据。比如很多表有 id 字段，id 可能有递增递减特征;还有的表具有时间戳 timestamp 字段,更是具有天然的单调可比较特性。

有了新旧数据的区分基准，我们就可以进行增量数据迁移设计。即每次数据迁移任务完成，都会记录本次数据源已经完成读取的位置，对本次数据迁移做记录。下次再迁移则会从已有的位置开始，对数据源进行读取迁移。结构如图：

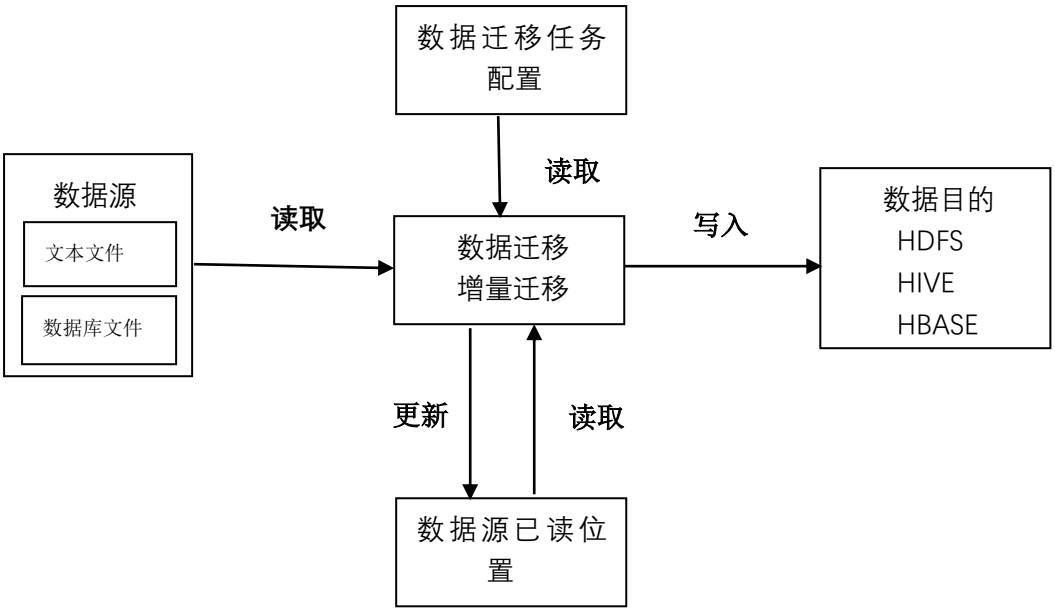

图 3.2 增量数据迁移结构图

3.3.3 流式数据迁移

在实际的企业应用场景中，数据源的数据往往处在实时变化之中。比如各服务器的日志文件在不断写入新内容，新的分析数据、报表数据不断写入业务数据库。而很多业务数据希望得到实时处理和分析。所以流式数据迁移或者是实时数据迁移是数据迁移云服务中的重要一个环节，也是最具挑战性的模块。

在流式计算环境中，数据是以元组为单位，以连续数据流的形态，持续地到达大数据流式计算平台。数据并不是一次全部可用，不能够一次得到全量数据，只能在不同的时间点，以增量的方式，逐步得到相应数据。

实时分析和处理数据流是至关重要的，在数据流中，其生命周期的时效性往往很短，数据的时间价值也更加重要。所有数据流到来后，均需要实时处理，并实时产生相应结果，进行反馈，所有的数据元组也仅会被处理一次。虽然部分数据可能以批量的形式被存储下来，但也只是为了满足后续其他场景下的应用需求。

数据流是无穷无尽的，只要有数据源在不断产生数据，数据流就会持续不断地到来。这也就需要流式计算系统永远在线运行，时刻准备接收和处理到来的数据流。在线运行是流式计算[19]系统的一个常态，一旦系统上线后，所有对该系统的调整和优化也将在在线环境中开展和完成。

流式数据处理如图：

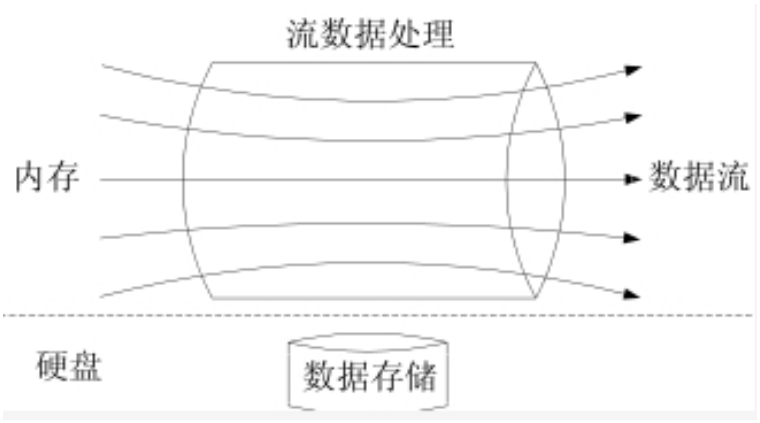

图 3.3 流式数据处理概念图

虽然数据源数据变化没有规律可循，可能某个时段涌入大量数据，而另外一个时段没有数据涌入。但作为流式数据迁移模块，可以周期性地对数据源进行检测。如果有新数据到来，则根据原有迁移任务配置创建新任务进行迁移，若没有新数据变化，则保持现有状态。如果把这个周期时间 T 设置的足够小，比如 10 秒或 5 秒，则近似地认为是实时数据迁移。流程图如下：

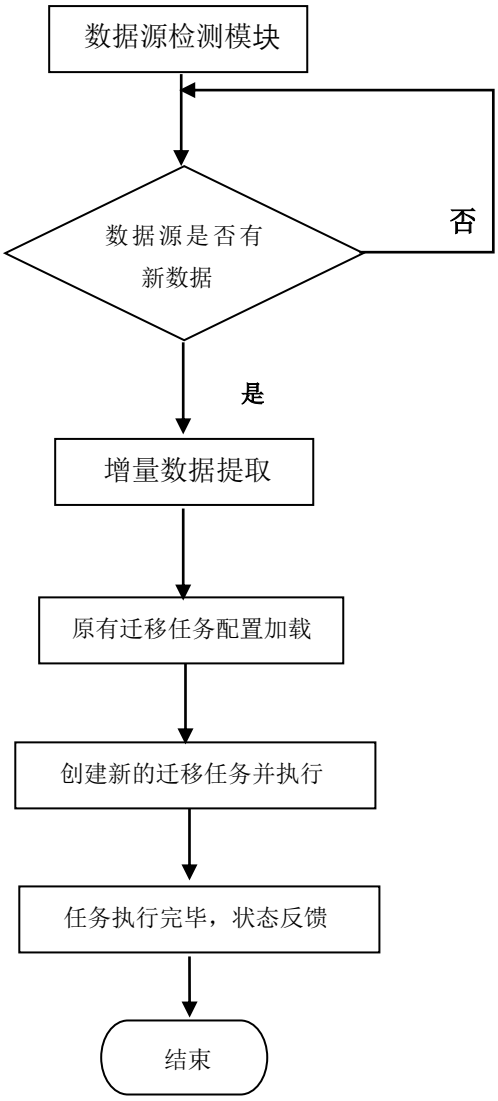

图 3.4 流式数据迁移流程图

3.4 流式数据提取优化

流式数据迁移对于增量数据提取有着更高的要求，设计行而有效的增量数据提取方案是流式数据迁移的至关重要的一步。对于文件类非结构化数据，仍可以以文件二进制位置作为新旧数据标准。而对于关系型数据库数据，仅以某个特殊字段如 id 或 timestamp 作为标准有着很大的局限性。因为对于数据源数据库数据表格结构我们不得而知，我们所设计的数据迁移云系统不能依赖于数据源，否则数据源数据库格式不同，我们的云服务系统则不能有效运行。那么这就是个失败的设计。针对关系型数据库设计通用的增量数据提取方案是必要和关键的。

关系型数据库数据增量提取方案采用数据库日志文件提取同步技术[20]。对

于 Mysql 这种非常流行的数据库，有 Binlog 文件实时记录着数据库数据的更改操作。Mysql 中的二进制日志文件主要有三种格式，一是 statement 格式，即每一条会修改数据的 sql 都会记录；而是 Row 行级日志，即日志中仅保存哪条记录被修改；三是 mixed 模式，即前两种的结合。由于 statement 级日志会对某些命令如”load data”造成数据不一致或丢失问题，而 Row 行级日志是最安全的日志方法，其记录的数据最全。它是以每行记录的修改来记录。因此通常情况下采用 row 级别日志，可以清晰看到数据库数据修改变化。当然根据实际业务特定需求，也可以调整为其他两种模式。

此时向 mysql 执行三条 sql 语句，如”insert into t1 values(2,’b’)”,”update t1 set name=’c’ where id=2”,”delete from t1 where id=1”。此时通过“show binlog events”命令查看基本日志信息如下图：

Pos	Event_type	Server_id	End_log_pos	Info
309	Query	1	381	BEGIN
381	Table_map	1	429	table_id: 79 (test.t1)
429	Write_rows	1	471	table_id: 79 flags: STMT_END_F
471	Xid	1	502	COMMIT /* xid=183 */
502	Query	1	574	BEGIN
574	Table_map	1	622	table_id: 79 (test.t1)
622	Update_rows	1	672	table_id: 79 flags: STMT_END_F
672	Xid	1	703	COMMIT /* xid=184 */
703	Query	1	775	BEGIN
775	Table_map	1	823	table_id: 79 (test.t1)
823	Delete_rows	1	865	table_id: 79 flags: STMT_END_F
865	Xid	1	896	COMMIT /* xid=185 */

图 3.5 Mysql row 级日志图

这里是基于 row 级别的二进制日志。记录着数据库数据变化的事务信息。Event_type 是这次数据库操作的类型，可以看到图中的 write_rows、update_rows、delete_rows 代表了之前的插入、更新、删除三种 sql 操作。Info 字段记录着数据变更的重要信息，主要是表明和 id 的映射关系，代表了数据变更的位置。这里的 pos 是个很关键属性。Pos 是记录本语句对应的日志的位置，对应二进制文本位置。这为数据库新旧数据判别提供了标准依据。即每次增量数

据提取，都会记录本次已读的日志文件位置，下次从新的位置开始读。

这里还看不出具体的 DML 语句和具体的数据变化。通过 mysql 自带的二进制转换解析工具可以看到如下内容：

```
# at 574
#160817 10:20:38 server id 1  end_log_pos 622 CRC32 0x9fa3cabc     Table_map: `test`.`t1` mapped to number 79

# at 622
#160817 10:20:38 server id 1  end_log_pos 672 CRC32 0xb1646398     Update_rows: table id 79 flags: STMT_END_F
### UPDATE `test`.`t1`
### WHERE
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='b' /* VARSTRING(30) meta=30 nullable=1 is_null=0 */
### SET
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='c' /* VARSTRING(30) meta=30 nullable=1 is_null=0 */

# at 672
#160817 10:20:38 server id 1  end_log_pos 703 CRC32 0x91a90c52     Xid = 184
COMMIT/*!*/;

# at 703
#160817 10:20:43 server id 1  end_log_pos 775 CRC32 0x5ae24c0b     Query    thread_id=12    exec_time=0    error_code=0
SET TIMESTAMP=1471400443/*!*/;
BEGIN
/*!*/;

# at 775
#160817 10:20:43 server id 1  end_log_pos 823 CRC32 0x33c52e84     Table_map: `test`.`t1` mapped to number 79

# at 823
#160817 10:20:43 server id 1  end_log_pos 865 CRC32 0x77e907a2     Delete_rows: table id 79 flags: STMT_END_F
### DELETE FROM `test`.`t1`
### WHERE
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
###   @2='a' /* VARSTRING(30) meta=30 nullable=1 is_null=0 */

# at 865
#160817 10:20:43 server id 1  end_log_pos 896 CRC32 0xb0988385     Xid = 185
COMMIT/*!*/;
```

图 3.6 Mysql Row 级日志详细图

从以上图中可以清楚地看到数据库数据变更和对应的 sql 语句。这里”# at”是对应日志二进制位置即 pos 数据。这里表明和列名都有相应的数字作映射。这里的列名并没有直接给出，因此在解析日志前需要根据表格结构做转换，详细实现在下章会有所论述。

数据源有流式数据到来后，会在备份数据库节点将这些信息同步作为一个新的数据源。这是考虑到数据迁移任务尽可能小的影响数据源。因为流式数据中，数据源的数据在不断变化，数据源在频繁的写入，短期内可能有高并发的写入。如果流式数据迁移选择直接从数据源数据库读增量数据，将会给数据源数据库带来更大的负载压力，将会影响到数据源性能甚至导致其宕机。

这里流式数据的提取，业内现有的一些迁移工具采用方案：将数据源增量数据同步至数据库备份节点。这里的备份是指将流式增量数据插入到备份节点数据库中，然后再从新数据库中读取数据进行迁移。如图

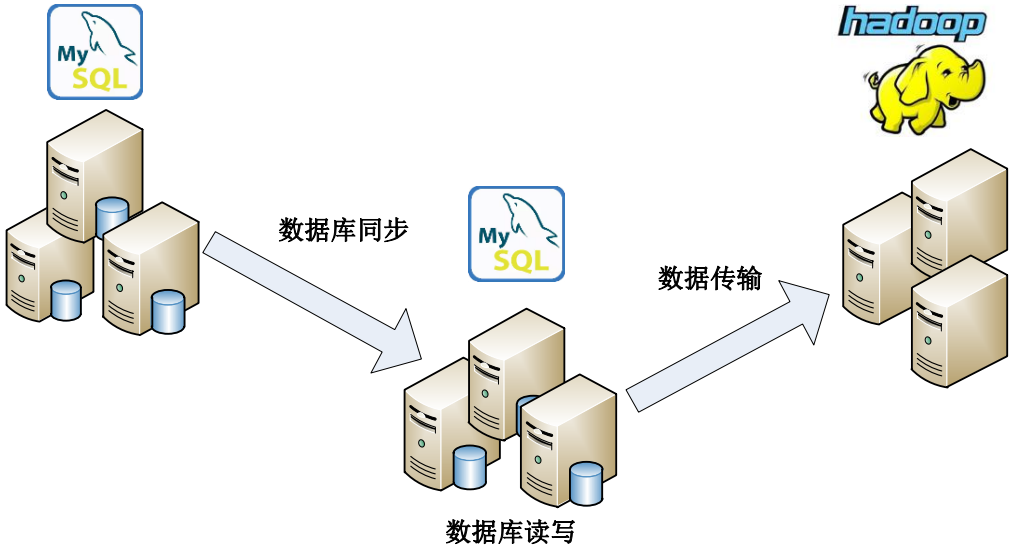

图 3.7 优化前流式数据提取架构图

本文所采取的方案是，将数据源日志操作文件同步到中介节点，将增量数据解析提取并并封装成为 event 事件传输给目标 Hadoop 集群。这样大大降低数据迁移对数据源负载性能的影响。方案选型如图：

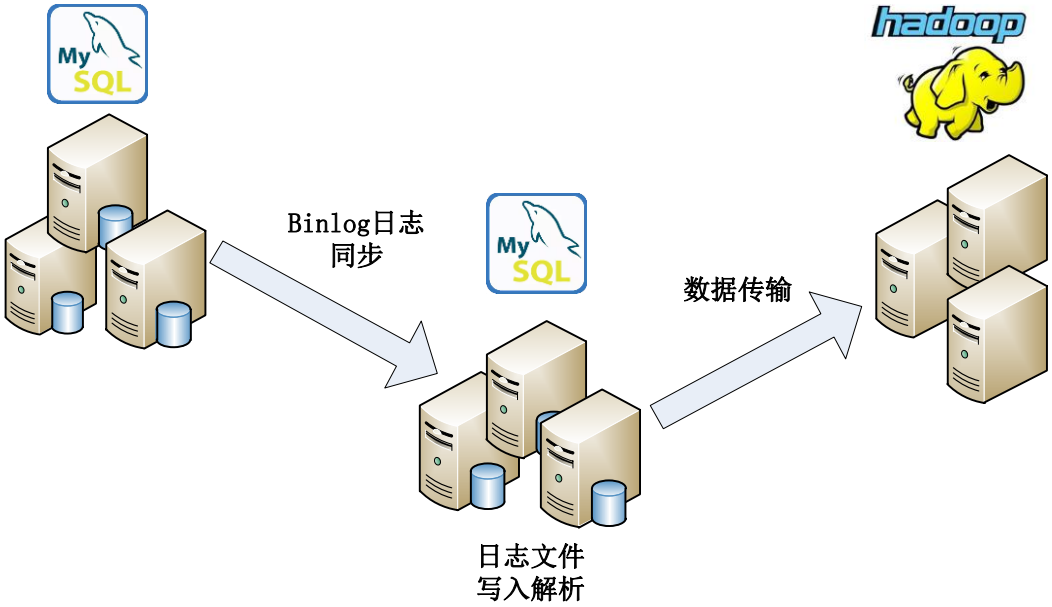

图 3.8 优化后流式数据提取架构图

优化前方案，是将增量数据同步至备份节点数据库，这里的开销是新备份数据库节点数据读写开销；而优化后方案，只同步 binlog 日志文件，在备份节点进行日志文件写入和数据解析，没有了备份节点数据库数据写入代价。优化前后

性能对比在第 6 章实验部分论述。

3.5 数据迁移云服务化设计

数据迁移系统在已有的工具和实现里往往是一种单机运行的软件，当业务部门人员有数据迁移需求时，会请求数据部门人员对其已有机器进行软件安装。安装完成之后，还要组织人员进行软件使用学习，这可能会耗费不少时间。软件使用的不当，还时常出现崩溃现象。因此单机数据迁移软件具有如下缺点：

(1) 繁琐的安装和不小的学习成本。

(2) 由于迁移依赖于单个节点，计算能力很有限，迁移效率不高。

(3) 几乎没有高可用性，没有故障恢复能力。

(4) 可视化效果很差。

本文所论述的数据迁移系统是基于云计算的一种数据服务。即数据迁移主体功能建立在云端集群，借助强大的分布式计算和存储能力实现更高效地数据迁移。迁移服务在云端而非本地，这也为远程操作数据迁移提供了可能和便捷。业务人员通过相应的 web 界面可远程查看迁移执行状态。同时云服务的设计也充分考虑了集群服务的高可用性。因此，数据迁移云服务化有如下有优点：

(1) 对于使用人员，无安装部署的烦恼。数据迁移此时作为一种公共云服务而存在，由专门的数据团队去维护和开发云，用户只需要通过 web 界面去申请、配置服务即可。几乎没有繁琐的软件使用学习。

(2) 简单易配置。用户无需关注 HDFS、Hive、Hbase 数据如何读写，无需关心集群配置。只需在本云服务提供的 web 端进行任务申请、任务资料补充即可执行数据迁移服务。

(3) 高可扩展性有助于强大的计算能力。数据迁移的主体建立在集群基础上，架构的设计充分考虑高可扩展性。比如，通过增加服务器节点能够提高集群迁移的吞吐量等性能。

(4) 高可用性。集群单个节点的故障不会对集群功能造成多大影响，通过心跳或分布式租约机制，单点故障能够得到快速检测并实现服务切换。集群可以实现 24 小时不间断服务。

(5) 有力的故障恢复机制。借助强大的监控和预警系统，对于数据迁移环节出现的故障，能够及时检测定位，并能够实现任务自恢复功能。

(6) 简洁鲜明的可视化系统。使得数据迁移的环节过程可控可视。如任务执行百分比、出错位置、异常信息等。

数据迁移云服务示意图如下：

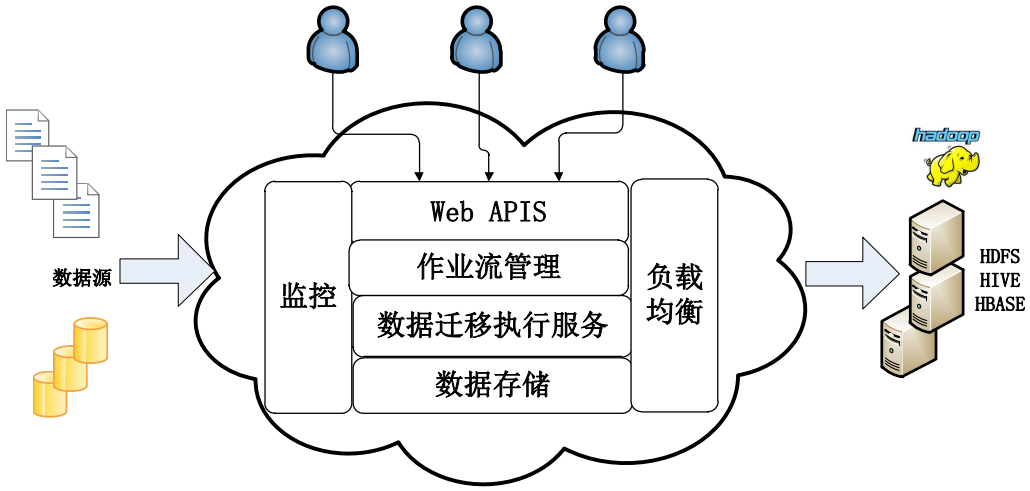

图 3.9 数据迁移云服务结构图

云服务从技术实现角度来说，主要是把基础服务以 B/S 的架构发布出去。对于公共基础服务，上层会有一层 web apis 层。用户通过调用 web 相应接口，实现对底层服务的调用。即用户通过浏览器去使用、查看后台服务。在实现阶段，系统使用了业内流行的 SPRING MVC 架构去组织各模块。MVC 全名是 Model View Controller,是模型、视图、控制器缩写。是一种架构设计典范，其高度重视解耦的意义。将业务逻辑、数据、界面分开考虑的设计思想融入其中。使得系统可扩展性和可维护性得到大大提高。模型负责封装应用程序数据在视图层的展示，视图仅仅只是展示数据，控制器用来负责接收用户请求。其核心思想是将业务逻辑从界面中分离出来，允许它们单独改变而不会相互影响。MVC[21]示意图如下：

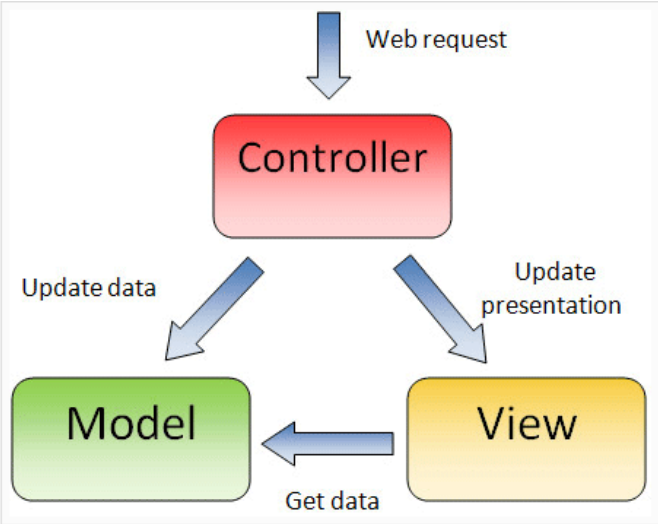

图 3.10 MVC 结构图

数据迁移云服务系统中，有以下结构层次：

- 云服务控制层。主要负责接收用户的请求，如迁移任务申请、执行、恢复、状态查看等。
- 云服务逻辑层。这里是云服务功能的核心，即各个业务逻辑实现模块。如流式数据提取逻辑、数据传输逻辑等等。这里也是可以扩展功能模块。
- 云服务数据持久化层。整个云服务有若干数据需要存储，比如迁移任务信息存储、监控数据储存等。储存单独为一层，有利于扩展和维护。
- 云服务公共技术层。这里的公共技术层可能是监控、报警等，是各个层共同需要的技术。
- 云服务视图层。这层主要组织数据的可视化。对于系统若干模块数据的格式化展示。

监控报警模块的设计与实现也是整个数据迁移云服务的重点之一。因为数据本身的价值是巨大的，数据是一种无价的资产，数据迁移本身是把宝贵的数据从一个地方高效、安全地传送到目的地。数据的迁移过程必须加以监控、跟踪、保护。一个没有任何监控报警的数据迁移系统是难以想象的。实际的分布式环境复杂和具有不确定性。如网络可能阻塞、迁移进程死掉、物理主机磁盘损坏等。在设计这个云服务时，必须要能够应对潜在的故障，对所有参与数据迁移的节点、进程、网络等采取必要的监控措施。使得出现故障可以追踪、出现严重故障能够

及时报警通知维护人员。

监控系统主要数据流图如下：

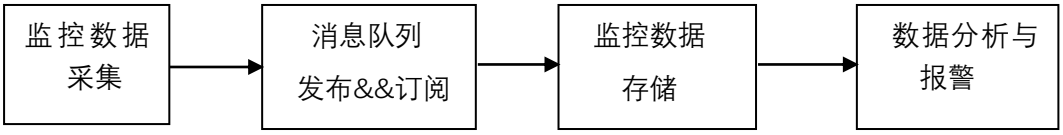

图 3.11 监控系统数据流图

监控系统设计目标和原则如下：

- 监控系统与是数据迁移云服务的一部分，要能够与现有的模块系统达到低耦合的状态。监控系统的设计要体现模块化、独立性，不依赖于具体的应用、组件、服务器。同时又基于已有的功能模块，其加入要尽可能降低对已有执行服务器性能和任务执行的影响。
- 对物理服务器节点的监控。云服务系统往往是很多节点在运行，云服务的性能提升往往可以通过增加服务器节点实现线性扩展。当服务器节点增加或移除，监控系统要能及时发现并通知业务人员。同时服务器的在线执行状态，要能及时获得和跟踪。这里主要是监控与执行节点保持心跳联系。如节点损坏，要能及时反应并处理。对于服务器当前的状态，CPU、IO 等状态指标进行实时跟进，并提供数据供可视化。
- 对执行进程的监控。这里的进程监控，也是对数据迁移一些重要模块的监控。如增量数据提取、数据过滤和解析、数据传输等。这也是本系统设计的一大特色，即数据迁移整个过程一定是可以追踪的。任何一个执行环节出现问题，都能及时反应和报警。比如增量数据监控进程死掉，要能及时重启，若重启仍有异常，要及时报警。
- 对监控数据指标进行有效地收集，并且高效存储供数据分析和可视化。监控的数据指标往往具有重要的分析和参考价值。一些数据需要进行持久化，无论是物理服务节点监控还是进程监控，要能够对原始监控数据进行统一格式化，有效快速地放入存储系统进行持久化。因为存储到数据库，可以查看历史监控数据、节点健康程度数据，同时能够通过数据挖掘和预测等手段，对云服务系统进行改进和提升。

- 对各类监控数据进行有效管理，并提供一个统一的可视化、管理、报警。要给业务人员一个统一的 web 界面，实时显示物理服务器、执行任务状态。根据用户配置的选项和参数，当相关指标达到用户设定的阈值时，及时进行报警。报警可通过多种手段，如邮件、短信。
- 监控系统要有助于实现集群的高可用性。集群上会跑很多数据迁移任务，会有多个用户在等待响应结果。对于重要的执行任务节点，实时监测，一旦出现异常，立即进行节点切换。把服务迁移到另外一个节点继续执行。尽最大可能保证系统能够持续性服务。
- 监控系统要有助于实现数据迁移故障恢复。数据迁移执行链路，从数据源，经过数据解析转换过滤，到数据目的地 hadoop 集群 HDFS、Hive、Hbase。在理想状态下，数据一路顺风到达目的地。而实际生产环境，可能会出现任务中断、异常退出，如数据源可能数据异常导致的执行任务异常中断。监控系统要能及时发现死掉的执行进程，及时重启进程并根据日志文件记录及时恢复迁移任务。对于重大故障要及时报警，让运维人员排查处理。

3.6 本章小结

本章主要介绍数据迁移云服务的总体设计和各模块设计。总体设计从企业实际需求出发，考虑集群设计的高可扩展和高可用性，并根据需求划分了若干模块，讲述了各模块所实现的功能和设计。

第4章 流式数据迁移设计与实现

4.1 流式数据迁移核心架构

数据迁移的核心在系统中称为 collector。该模块设计要考虑该节点的状态管理、数据库日志抽取与解析、数据传输等。

架构图如下：

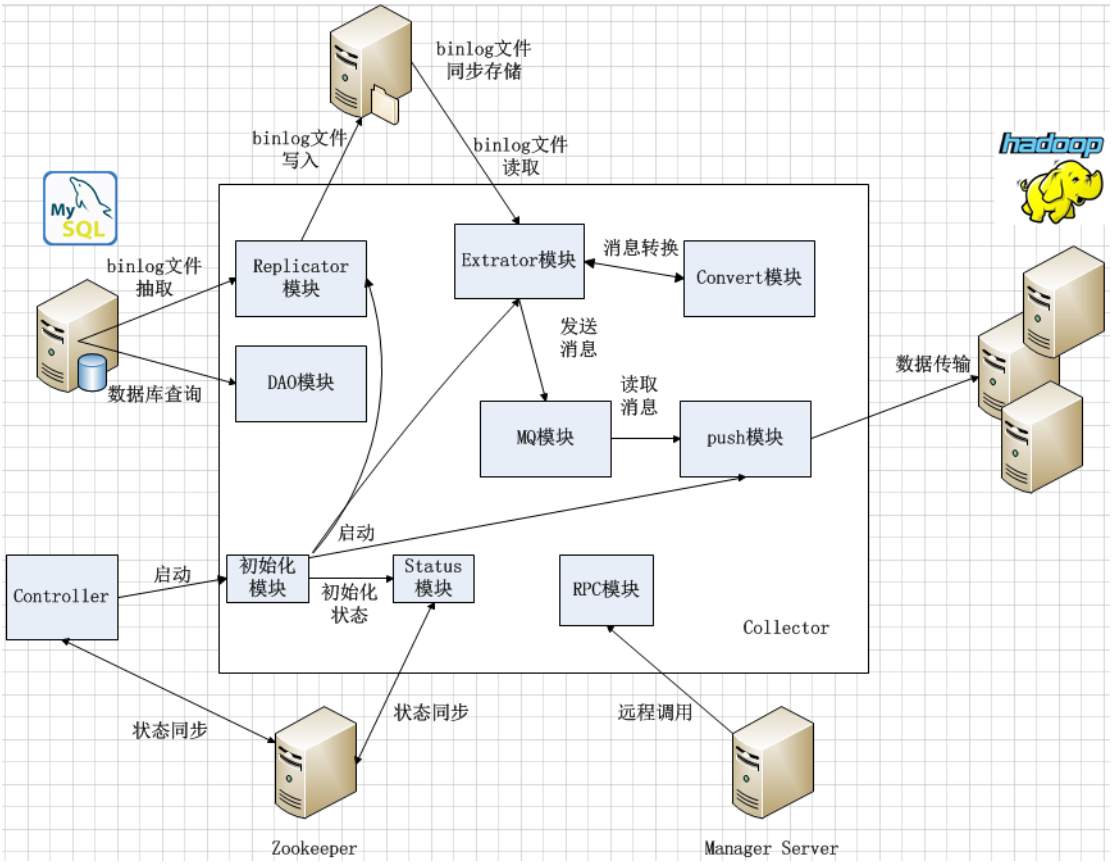

图 4.1 数据迁移核心架构图

如图所示。这里用 zookeeper 做集群协调和管理服务，如消息分发、状态一致性同步、节点注册等分布式服务。远程调用模块用于控制 collector 整体执行状态等。关系型数据库日志文件经过 Replicator 抽取，然后写入指定的 binlog 文件存储服务器。Extrator 模块会抽取相应的日志文件。Binlog 日志文件会通过 convert 模块做相应转换，即根据迁移任务配置模块，把相应的数据抽取出来

并转换为相应的 event 事件消息。该消息通过消息队列、push 模块传输到目的地 hadoop 集群。

4.2 集群初始化

由于集群是在 zookeeper 监控下运行，迁移执行服务节点需要向 zookeeper 做一些注册任务以及配置加载任务。初始化模块的输入是在 zookeeper 上注册的 id，输出是返回信息。如启动成功，则 status=running,若启动失败则 status=stop，返回错误信息如“启动失败，请检查相关配置”。

初始化模块流程如下：

（1） 初始化连接配置。

（2） 加载配置：根据收集器 id 从 zookeeper 相应位置处获取配置，包括

- 路径配置：数据收集器主目录路径配置；relayLog 路径配置，该路径用于存放 mysql binlog 备份日志路径。
- 监听的数据源数据库节点。采用 xml 格式描述，如:

```
<datasource id=<datasource-id>>
    <db-driver>db-driverClass</driver>
    <url>url</url>
    <username>username</username>
    <password>password</password>
</datasource>
```

监听的数据库表。

```
<ResourceTable>
    <TableName>table1</TableName>
     …
    <TableName>table2</TableName>
</ResourceTable>
```

- Filter 配置。可以是简单的 sql 的 where 条件，或是其他过滤配置，可以后期扩充。

（3） 将数据收集模块状态设置为 INITIATE。

（4） 加载其他子模块。如数据库日志复制模块、数据抽取模块等。

（5） 如果一切成功，则将状态设为 RUNNING，如果发生异常则将状态设为 STOP，并通知报警。

4.3 流式数据提取

实际企业业务中，数据以流的形式源源不断地涌入关系型数据库。流式数据提取是实现流式数据迁移的核心。根据上一章总体设计分析，可以通过复制数据库日志、周期性地解析数据库日志、从日志文件中抽取增量数据来实现该模块。

流式数据提取概要步骤如下：

图 4.2 流式数据提取概要步骤图

流式数据提取模块主要有两个子模块，一是关系型数据库日志文件复制模块 Replicator，一是数据抽取模块 extractor。

4.3.1 数据库日志文件复制

Replicator 模块输入是数据源信息和新 log 存储位置，输出是在传输服务节点指定位置产生数据库操作日志文件，如 mysql relay log。

Replicator 模块工作流程(以 mysql 增量数据提取为例):

(1) 根据 mysql 协议监听 mysql 节点的 binlog 数据变化。

(2) 将指定数据源的 binlog 二进制日志文件数据写入执行服务节点本地 relay log 中。

(3) 记录下每次读取数据源 Binlog 位置，这是流式数据迁移已完成工作量的标志。该标志可用于故障恢复、断点恢复。比如执行迁移服务节点因不可预测原因宕机，重启不需要从头读取所有日志内容，而是从上次已读取的地方继续读取。

(4) 每次读取一个 binlog 信息块检查下 Replicator 状态位，如果是暂停，比如此时用户希望暂停任务，则阻塞当前线程，等待 RPC 模块主动唤醒线程继续运行。

Replicator 模块流程图如下：

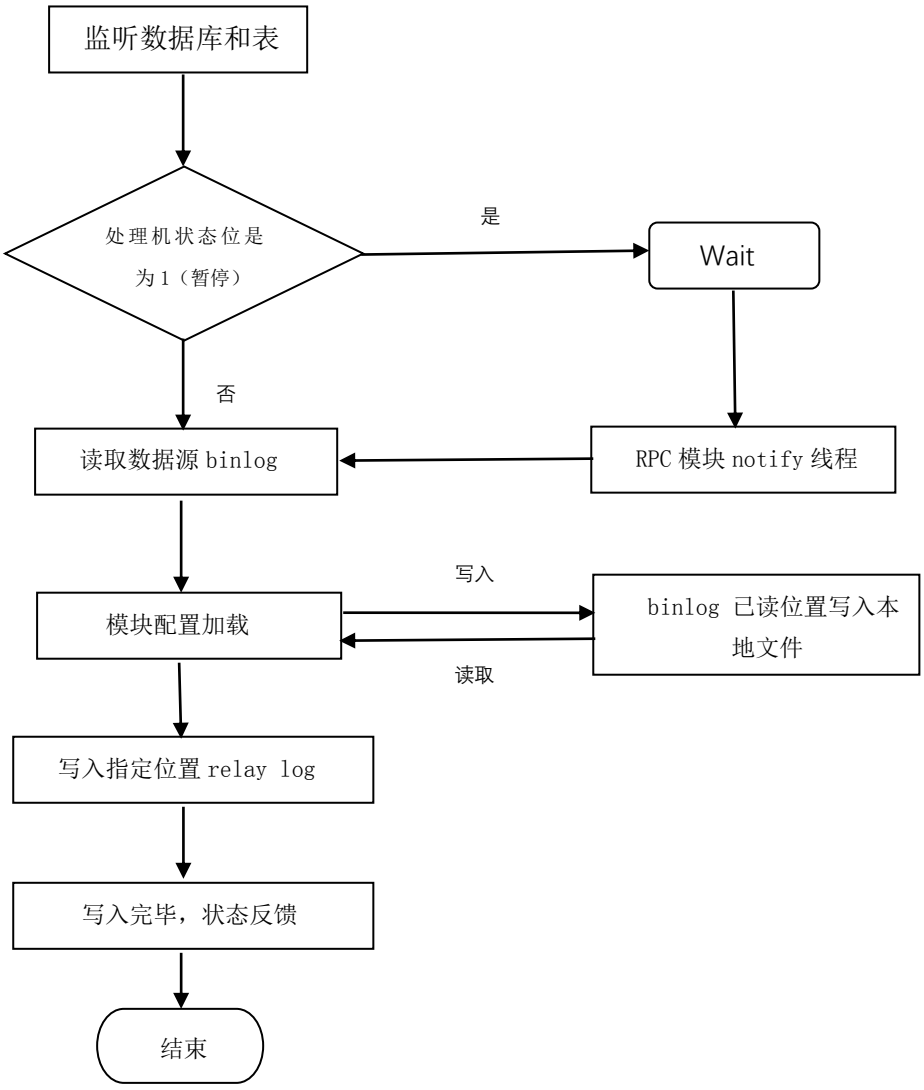

图 4.3 Replicator 模块流程图

该模块类图如下：

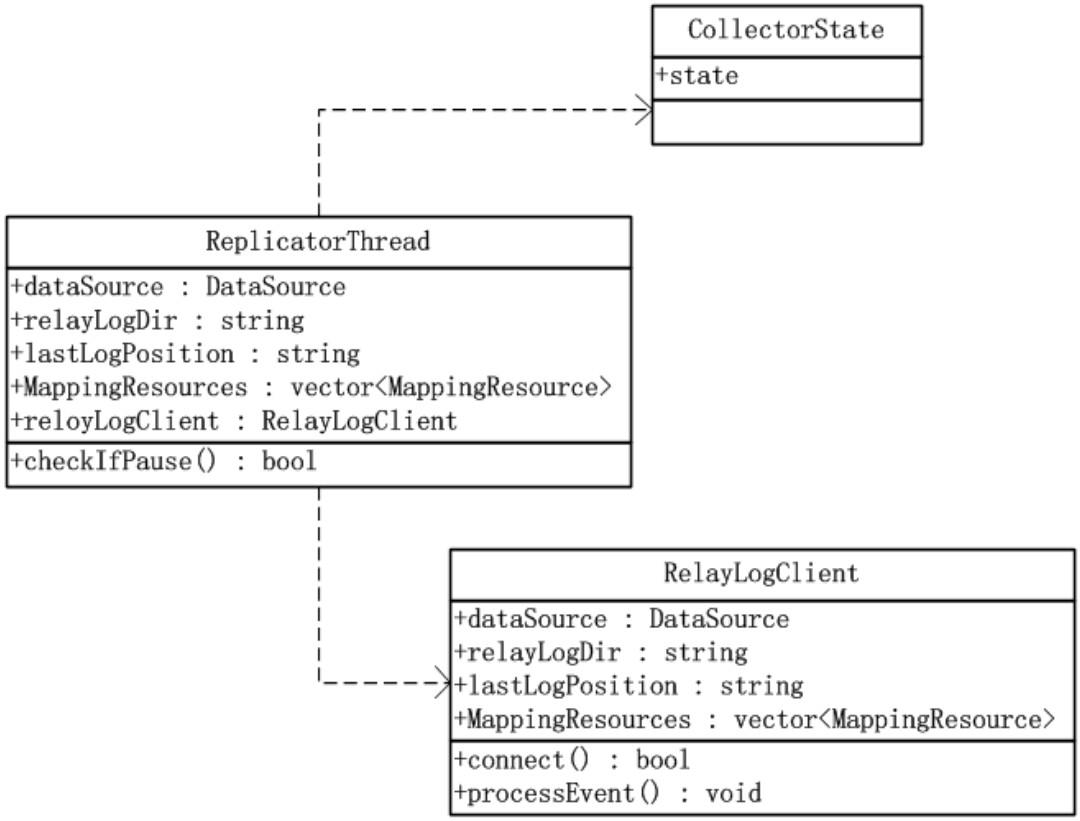

图 4.4 Replicator 模块类图

4.3.2 日志文件解析与数据提取

Extrator 模块负责解析数据库日志文件 relay log，并把需要的数据信息生成 event 事件供传输迁移。输入是 relay log，输出是 message 消息对象，用于 push 模块传送。其主要流程如下：

（1） 初始化表元数据信息

- 根据 datasource 配置连接 mysql。
- 获取 mappingResources 中配置的表。
- 依次对每个表调用 DAO 模块的 descTable 方法，获取表的列名与序列号的映射关系。该元信息在 convert 模块生成的消息对象时使用。

（2） 根据 binlog 二进制文件格式解析 relay log 文件。

（3） 当遇到 alter table 事件时，调用 DAO 模块的 descTable 方法，获取更新后的表的列名与列序列号的映射关系。

（4） 将 binlog RBR(Row-based replication)事件转化为 event 对象。

- 将非事务的 RBR 事件转化为一个 event 对象，data 属性包含该事件的数据：
  - TableName
  - Schema
  - 事件类型
  - 表的列数
  - 各列的类型
  - Column values:修改后的数据，只在 INSERT,UPDATE 事件中存在，按序列号排列。
  - Key values:修改前的数据，只在 DELETE，UPDATE 事件中存在，按序列号排列
- 将一个事务内的多个 RBR 事件转化为一个 event 对象，data 属性是该多个事件的数组。

RBR binlog 包含的内容：

- Timestamp: statement 开始执行的时间，精确到秒。
- Server_id: 产生日志的 mysql 服务器的 ID。
- Eventide: 包含 serverId,binlog 文件和该 event 的 offset，该 eventId 可以作为 binlog position 使用。
- Binlog version:1,3,4
- Create timestamp: event 日志产生的时间，精确到秒。
- Event_type: event 类型。
- Data: event 内容，会根据 event 类型的不用而不同。

RBR 事件[21]类型：

- 事务开启日志：类型为 QUERY_LOG_EVNET，event 内容为：sql operation 类型的为 BEGIN 开始的数据。
- TABLE_MAP_EVENT：只在 RBR 日志格式中使用，event 内容为 tableName 操作的表明;schema 操作的 schema;event_type 事件类型;column count 列数;column types 各列类型。

- ROW_LOG_EVENT:

  WRITE_ROW_LOG_EVENT:column spec 为发生操作的序列号，可能有部分列会不出现，如[1,2,3…];column values 为发生操作的行值，根据 column spec 顺序排列。

  DELETE_ROW_LOG_EVENT：key spec 为删除的数据的列序号，可能有部分列会不出现，例如[,2,3…];key values 为删除操作前的行值，按 key spec 顺序排列

  UPDATE_ROW_LOG_EVENT:column spec 为老数据的列序号，可能有部分列会不出现，例如[1,2,3…];column values 为老数据的值，按 column spec 排列;key spec 为修改后的数据列序号;key values 为修改后的数据的值，按 key spec 排列。

- XID_LOG_EVENT 为事务提交的日志，event 内容为事务的 xid。事务的封装也就是以 QUERY_LOG_EVENT 事件并且 sql operation 为 begin 代表事务开始，以 XID_LOG_EVENT 代表事务的结尾，中间的所有事件为一整个事务。

（5） 将 event 对象交给 convert 模块处理，转化为消息对象。

（6） 将消息对象放入 push 模块的等待队列中。

（7） 每将一个消息对象放入 push 模块后检查状态位 ColletorState.EXTRACTION_STATUS 是否为 1，如果是则暂停当前 extractor 线程，等待 RPC 模块主动唤醒线程继续运行。

流程图如下：

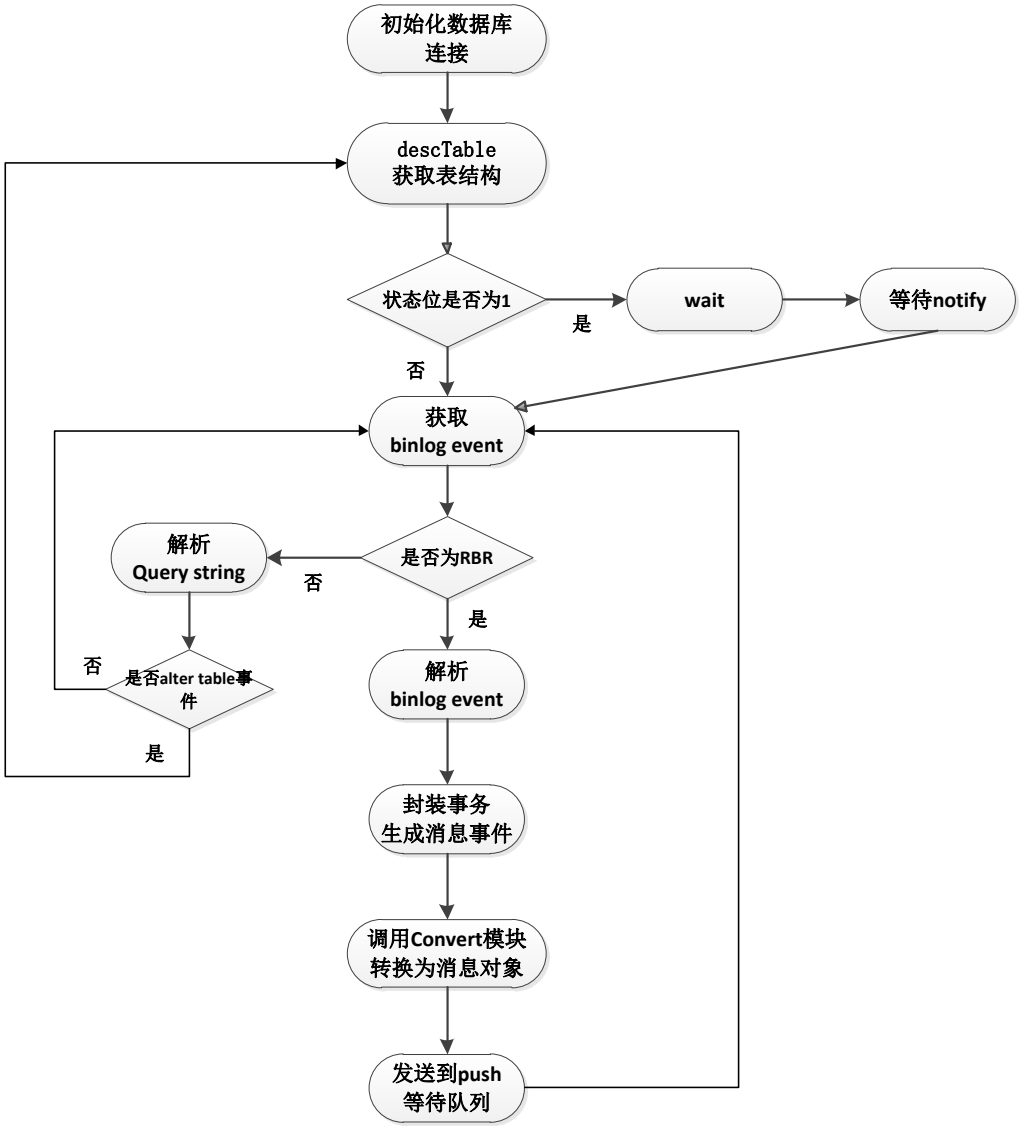

图 4.5 Extrator 模块流程图

Convert 模块是一个独立模块，会被 extractor 模块调用，处理 extractor 模块转化后的 event 对象为消息对象。

具体功能如下：

（1） 初始化功能：在 collector 启动时要先初始化 convert 模块，需要做如下工作：

- 加载 TOPIC 配置信息，生成 topic 对象模板

```
Topic1：{
    Prop1:null,
```

```
    Prop2:null,
    …
}
```

- 加载 ORM 配置信息，生成各个表的映射对象

```
{
    TableName:”table1”;
    Id:[“col1”], //表主键
    Fileds:
        [{
            ColName=”col1”,
            [  //列映射配置
                {
                    Topic:$topic1,
                    Prop:$prop1,
                },
                {
                    Topic:$topic2,
                    Prop:$prop2
                },
                …
            ]
        }]
}
```

- 加载 extractor 模块获取的表元信息

```
Table1:{
    1: col_name_1, //序号与列名的映射关系
    2: col_name_2,
    …
}
```

（2） Convert 功能：

提取 event 对象中的 data 属性，遍历该数组中的每个 data 对象，根据事件类型分别处理：INSERT 提取 key values;UPDATE 提取 key values；DELETE 提取 column values。从而获得了一个数组对象，内

容为按序列号排列的值。

根据表元信息，生成列名和值映射的对象结构，格式如:table1{col_name1:value1,col_name2:value2}。

再根据 ORM 配置中的列映射关系，构造出相对应的 topic 对象，格式如:

```
topic_name:{
      action:update,//时间类型
      prop1:value1, //topic 属性
      prop2:value2
      …
}
```

有一个事件类型属于特例：DELETED。当事件类型为 DELETED 时，不需要构造 topic 的其他数据，只需要获取主键数据即可。

当配置 topic 的 needFullData 为 true 时，需要反查 topic 的其他关联属性。通过以上步骤将单个 event 中的对象化成了一个 topic 对象，一个 event 对象包含多个 data 数据，还需要一个 event 中所有的 data 数据生成的 topic 对象做一个整合，整合后生成一个消息对象，格式为 JSON。示例如下

```
{
   Topic:<topic_name>,
   Timestamp:event 发生的时间
   Events:[
      {
          Action:INSERT|UPDATE|DELETE,
          Data:{
             $ property1:value1,
             $ property2:value2,
             …
          }
      },
      …
   ]
}
```

该 JSON 对象第一层为公共信息:topic 表示该消息对象属于哪个

topic;timestamp 表示该消息对象发生的时间，类型为 long;events 表示一个数组，每个数组都是一个 RBR 事件生成的 topic。

TemplateParser：用于解析 ORM 配置文件，生成 Table 对象存储映射关系。Converter 用于将 DBMSData 对象转化为 Message 对象。Message 用于发送到消息队列中的对象。TopicData 用于存储消息数据的对象，TopicData 为一个抽象父类，不同的 topic 会构造不同的子类对象。

DAO 模块用于数据库操作，主要有以下功能：

查询表结构。接口如

```
List<Column> descTable(String schema,String tableName);
```

根据主键进行单表查询。接口如

```
HashMap<String,Object> queryTableForObject
(String schema, String tableName, LinkedHashMap<String,Object>  keyvalues,
List<String> returnFields )
```

4.4 数据传输

数据的传输这里会利用开源 flume[22]系统组件做可靠性传输。Flume 具有着分布式、高可靠、高可用性等特点，在大数据数传输领域里经常使用。

上面一小节，论述了从数据库日志文件提取数据，并封装成 event 事件消息发送， Event 事件本身为一个二进制字节流数组，event 也是 flume 数据传输的基本单位。代表着数据流的最小完整单位，从数据源流向数据目的地。

传输系统的核心由三部分组成，数据传输数据流如下：

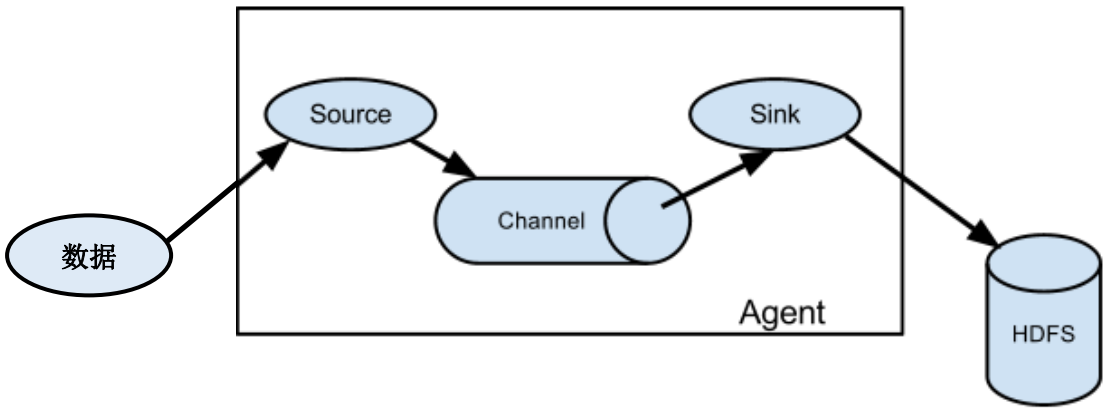

图 4.6 数据传输数据流图

Source[23]部分用于接受外部传来的消息数据。如某个监听文本文件数据，数据库增量数据等。

Channel[24]类似一个存储池，接受 source 的数据的输入，并做一定缓冲，然后数据流入 sink。Channel 常见的有三种类型。MemoryChannel 用于使用的内存通道，可以实现高吞吐，但是无法保证数据的完整性。且如果遇到服务器断点或死机，数据可能会部分丢失；MemoryRecoverChannel 是老的模式，在官网上已建议由 FileChannel 来替换；FileChannel 是利用了文件通路，即持久化所有事件到磁盘。即使当节点挂掉，仍可以从文件中恢复数据。因此这种模式可以保证数据的完整性与一致性。但是相应的性能会有所降低。

Sink[25]则是数据流导向。可以向文件系统、数据库、hadoop 存数据。

数据传输的可靠性，主要由这样的机制保证：数据在送到目的地之前，会先缓存起来，待数据真正到底目的地后，才自动删除缓存数据。Flume 使用事务性的方式保证传送 Event 整个过程的可靠性。Sink 必须在 Event 被存入 Channel 后，或者，已经被传达到下一站 agent 里，又或者，已经被存入外部数据目的地之后，才能把 Event 从 Channel 中 remove 掉。这样数据流里的 event 无论是在一个 agent 里还是多个 agent 之间流转，都能保证可靠，因为以上的事务保证了 event 会被成功存储起来。而 Channel 的多种实现在可恢复性上有不同的保证。也保证了 event 不同程度的可靠性。比如 Flume 支持在本地保存一份文件 channel 作为备份，而 memory channel 将 event 存在内存 queue 里，速度快，但丢失的话无法恢复。

4.5 状态管理与远程调用

Status 模块管理 controller 在 zookeeper 上的状态。提供的功能有设置 collector 状态如 1-INITIATE(collector 正在初始化);2-RUNNING(collector 正在运行);3-STOPPING(collector 正在停止);4-STOPPED(collector 已停止)。记录错误日志功能，在 collector 任何模块发生错误记录错误信息，连接 zookeeper 记录在相应位置。

RPC 模块负责远程调用[26]，实现对 collector 节点的控制。其主要功能有:

(1) 查询模块运行状态。

表 4.1 RPC 查询模块表

Key	必须	类型	备注
action	是	字符串	QueryStatus
models	是	int	有 4 个 bit 组成 第一位表示是否需要 collector 状态信息 第二位表示是否需要复制日志状态信息 第三位表示是否需要抽取消息状态信息 第四位表示是否需要推送消息状态信息

返回的 JSON 如下：

```
{
   Connector:{  //collector 模块
     Status:RUNNING,     //connector 状态
     Topics:topic1,topic2    //当前处理的 topic，每个 topic 名称
     Channel:channel,        //发送的 channel
     Id:xxxx,            //connectorID
     StartTime:2016-12-08  //开始运行时间
   },
   Replicator：{ //复制模块
     RepTime:2016-12-08, //复制日志的最新时间
     Reposition:mysql-bin.000013:0, //复制日志的最新位置
     Status:RUNNING //复制模块状态
   },
   Extractor:{
     Status:RUNNING //抽取模块状态
   }
}
```

(2) 操作模块状态。

表 4.2 操作模块状态

Key	必须	类型	备注
Action	是	字符串	OperationModel
Models	是	Int	由 3 个 bit 组成 第一位表示操作复制模块 第二位表示操作抽取消息模块 第三位表示操作推送模块
Type	是	int	0- 恢复运行 1- 暂停

响应：{

Code:200 //响应码

ErrorMessage:xxxx //错误描述

}

(3) 停止 collector。

表 4.3 停止 collector

Key	必须	类型	备注
Action	是	字符串	stopCollector

错误码定义:

表 4.4 错误码定义

code	含义
200	请求执行成功
201	部分执行成功
400	请求错误(参数缺失、格式错误、值非法等)
500	服务器内部错误
503	服务不可用

4.6 本章小结

本章主要介绍了流式数据迁移的设计与实现。流式数据迁移核心问题在于高效提取流式数据，本文提出了基于数据库日志解析抽取数据的方案。该方案相对于先进行数据库同步的传统迁移方案，具有更高的流式数据提取效率和更低的计算资源占用，同时对数据源影响很小。

第5章 数据迁移云服务化设计与实现

5.1 集群管理

云服务所涉及到的集群节点数众多，这些节点也是按照功能模块划分不同的区。这些节点中必须选出一个作为主节点，即 master 节点。该节点负责整个云服务集群各个模块的启动和初始化工作、任务分发等工作。运维人员可以通过总控节点与整个集群交互，总控 master 节点存储着整个集群最重要的一些配置信息。

多个节点选出一个总控节点 master 节点涉及到分布式一致性问题。Paxos 协议[27]通常用来解决这一问题。如果能够保证多个节点操作的一致性，就能够选出唯一的 master 总控节点。其应用场景还有实现分布式锁服务、全局命名和配置服务等。Paxos 算法主要有两种角色，proposer 提议者和 acceptor 决策接受者。

一般情况下，云服务可能只有一个提议者 proposer,那么这种情况他的提议能够很快被多数节点接受。其基本流程包括两个过程：批准阶段，Proposer 发送相关信息要求所有其他节点接受这个提议值，acceptor 可以选择接受或拒绝；确认阶段，如果超过一半的 acceptor 接受，则提议值生效，这时 proposer 发送 ack 确认消息通知所有的 acceptor 提议生效。

但当网络出现异常时，云服务中可能有多个 proposer，而他们的提议又不相同。这时需要进行完整意义的选举算法[28]。具体算法流程如下：

（1） Prepare 阶段：Proposer 首先选择一个提议序号 N 发送给其他 acceptor 节点。节点收到该消息后，如果提议序号大于它已收到的所有序号，则 acceptor 将自己上次认同的提议恢复给 proposer，并且约定不再回复小于 N 的提议。

（2） Accept 阶段：此阶段为批准阶段。若在之前的 prepare 阶段 acceptor 回复了上次接受的提议，则 proposer 选择其中序号最大的提议值发给 acceptor 批准；否则，proposer 生成一个新的提议值发给 acceptor 批准。Acceptor 在不违法它之前约定的情况下接受这个请求。

（3） ACK 阶段：若超过一半的 acceptor 接受，那么提议值立即生效。

Proposer 发送确认消息通知所有的 acceptor 提议生效。

该算法要求每个生效的提议都被大多数节点接受，并且每个节点不可能接受两个提议值，这就保证了最终只有一个提议值得到接受，具有唯一性；同时，从算法执行过程看，只要有超过一半节点接受了提议，则 proposer 能够很快发现并裁决。所以随着算法的不断运行，它会不断向某一目标值靠拢。这就保证了算法一定可以终止并得出结果。

5.2 基于因子分析和响应时间的负载均衡算法

由于集群中有着众多的迁移任务在运行，为了提高集群的吞吐量和减少任务实际执行时间，良好的负载均衡策略是非常有必要的。前文中也分析了现有常见的负载均衡算法，常见的静态负载均衡算法是最朴素简单的算法，但是往往运行效果并不理想，没有考虑实际负载状态的动态变化。而常见的动态负载均衡策略如最小连接调度算法 LCS、最小权值连接调度算法 WLCS 等，虽然考虑了一些动态策略，但是考虑的负载因素还不够，还有着改善的空间。如任务的响应时间也是负载策略应该考虑的一个重要因素。

某个时刻对于某个服务器节点的负载情况考虑因素包括 CPU 利用率、内存、磁盘网络四个因素。论文中所提出的负载均衡算法将把任务执行时间也考虑进去。因为结合数据迁移任务的执行特点，有的服务节点尽管 CPU 利用率高，比如此时有一定的数据解析任务、启动的线程数多、MR 任务中的 Map 数多，然而此任务需要传输的数据量可能较小。比如流式数据迁移中，每隔周期 T 时间就会创建一个迁移小任务，这种增量迁移任务量相比较全量数据迁移，数据量要小得多。因此，这种小任务可能 CPU 利用率高但是执行时间很短。即高负载可能只是一个短暂的现象，任务很快执行完节点又处于低负载的情况。因此，选择执行服务节点时，如果 A 服务器上的任务执行时间与 B 服务器上的任务执行时间相差很大时，应该优先考虑任务执行时间很少的服务器，因为其很快会恢复低负载情况。这里用表达式去表示以上的想法，并逐步优化。

设服务器集合 $M=\{M_1,M_2,M_3,M_4\cdots M_N\}$ $(N>1)$，$W(M_i)$代表服务器的权值，$Q(M_i)$代表 M_i 节点当前任务排队数，$T(M_i)$代表 M_i 节点当前任务执行时间估计，$\lambda(M_i)$表示 M_i 节点当前的参数因子，且应该是动态变化的。得到节点负载情况 $C(M_i)$基本公

式如下：

$$C(Mi) = T(Mi) * \frac{Q(Mi)}{W(Mi)*\lambda(Mi)} \quad \text{公式 (4.1)}$$

$C(M_i)$代表了 M_i 节点当前负载大小的描述。因此每次任务负载时选择 $\min(C(M_i))$ $(1<=i<=N)$的节点进行任务分发。然而这里的$\lambda(M_i)$还没有确定，它代表了某些潜在负载因素所构成的参数。这里基于因子分析法用于负载均衡算法设计去构造参数$\lambda(M_i)$。

$L(M_i)$作为负载指数应该包含一些实时可以测量的负载值，用 C_i 代表第 i 种影响负载能力的变量，比如 C_1 代表 CPU 利用率、C_2 代表内存利用率、C_3 代表磁盘利用率、C_4 代表磁盘利用率。则可基本表示为$L(M_i)=f(C_1, \cdots, C_4)$。f 表示了某种关系函数。为了求解这个函数，采用因子分析法。首先建立正交因子分析模型。对于 C_i 有矩阵模型如下：

$$C = XF + \varepsilon \quad \text{公式 (4.2)}$$

这里的 F 代表了公共因子，这些因子不可测量但又是影响负载因子的潜在参数，ε 是公式中的特殊因子，X 是系数矩阵。他们的线性矩阵和决定着 C_i 可以测量的负载变量。由于 F 不可测量，这里可以用 C 来表示 F，用可测量变量去表示不可测量变量。这里 F 可以估算为

$$F \approx [(x^T x)^{-1} x^T]C = \beta^T C \quad \text{公式 (4.3)}$$

其中 β 为因子得分系数矩阵。把 F 换成 C，得到：

$$L(M_i) = f(\beta_1^T C_1, \cdots, \beta_4^T C_4) \quad \text{公式 (4.4)}$$

最终整理可得：

$$L(M_i) = \sum_{i=1}^{4} \mu_i \beta_i^T C + \theta \quad \text{公式 (4.5)}$$

μ_i为 F_i 的权重值，β_i 是前面的因子系数矩阵列向量，C 为可以测量的负载能

力变量。θ为经验值修正，防止负值。

以上因子分析法意在用定量计算式法确立负载指数。回到公式 4.1，对于$\lambda(M_i)$希望能根据节点负载情况动态调整。如果该节点的负载远高于集群平均负载，则认为其负载偏高，应该降低其$\lambda(M_i)$；若远低于集群平均负载，则认为其负载过低，可以分发更多的数据迁移任务，应该增加其$\lambda(M_i)$；否则可以保持不变。

综合以上分析，本文所提出的基于因子分析和响应时间[29]的负载均衡算法(Dynamic Load-balance Based on Response Time and Factor Analysis, DLBRTFA)。DLBRTFA 算法流程如下：

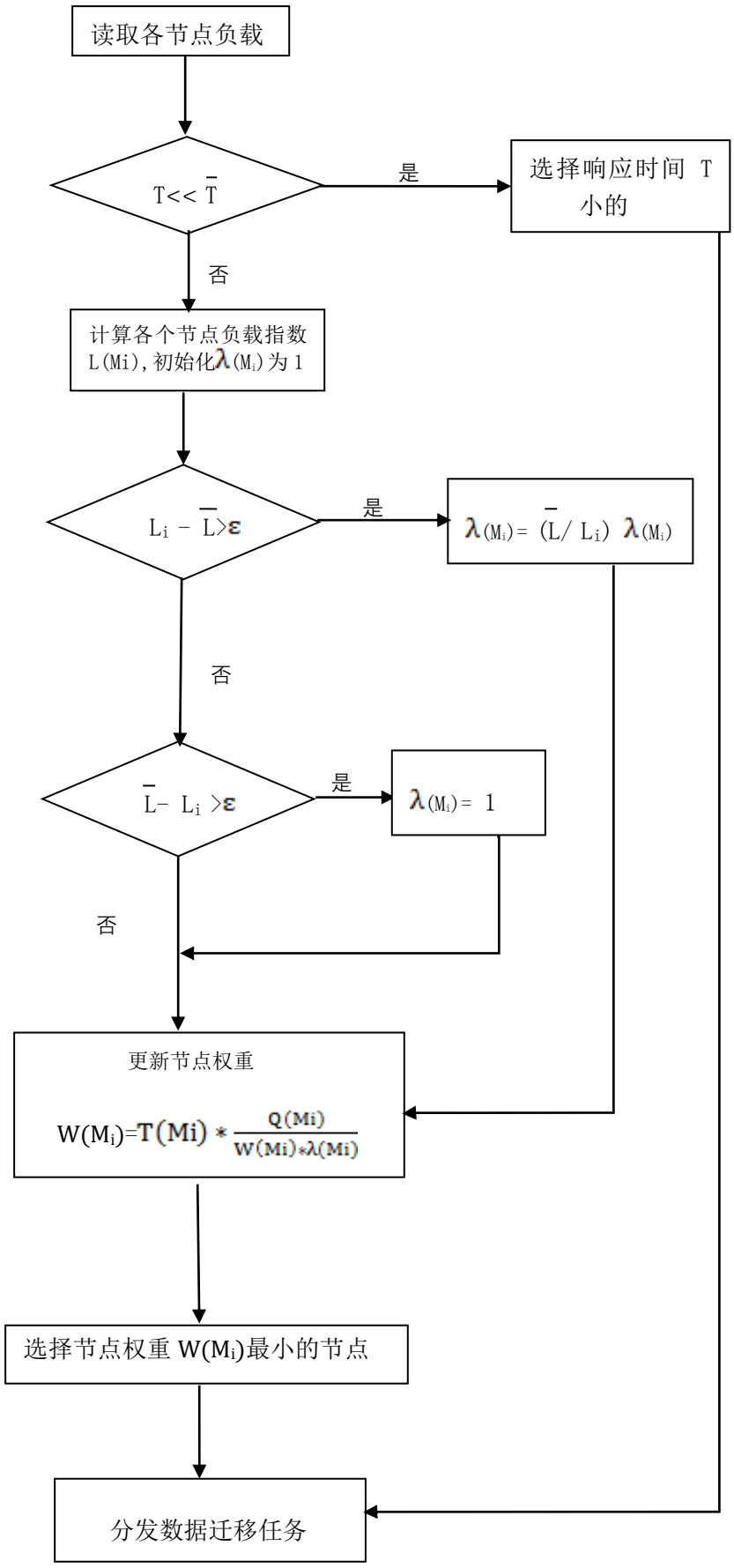
读取各节点负载
T<< $\bar{T}$
是
选择响应时间 T 小的
否
计算各个节点负载指数 L(Mi),初始化λ(Mi)为 1
$L_i - \bar{L} > \varepsilon$
是
$\lambda(M_i) = (\bar{L} / L_i)\ \lambda(M_i)$
否
$\bar{L} - L_i > \varepsilon$
是
$\lambda(M_i) = 1$
否
更新节点权重
$W(M_i) = T(Mi) * \frac{Q(Mi)}{W(Mi) * \lambda(Mi)}$
选择节点权重 W(Mi)最小的节点
分发数据迁移任务

图 5.1 DLBRTFA 算法流程图

5.3 作业流管理

数据迁移在实际企业的业务中，常常有这样的场景：有一个大的任务，这个任务可以划分成多个较小的任务完成，之所以进行划分是因为小任务之间可以并发的进行。如一个数据迁移任务，有数据提取、数据转换、数据加载、状态反馈、报表反馈等子任务。假设大任务 A 可以划分为 B、C、D、E 四个子任务完成，而 B 和 C 是可以同时进行的，D 依赖 B 和 C 的输出，E 又依赖于 D 的输出。于是我们一般的做法可能就是开两个进程同时执行 B 和 C，等两个都执行完成之后再执行 D，接着执行 D 和 E。整个执行的过程都需要我们参与，但是整个执行过程类似一个有向无环图，每一个子任务的执行可以看作整个任务的一个流，我们可以同时从没有入度的节点开始，任何没有流向（两个节点之间没有通路）关系节点都可以并行地执行。另外一种情况，迁移任务之间也可能有着先后或者依赖关系。比如同时有两个迁移任务 A 和 B 同时提交给云服务系统，而业务部门希望 A 迁移任务先执行，因为它的优先级可能较高，或者 B 的数据迁移任务需要依赖于 A 的执行。

当数据迁移任务之间有着作业流次序关系时，我们通过 DAG 图去表示它。如图：

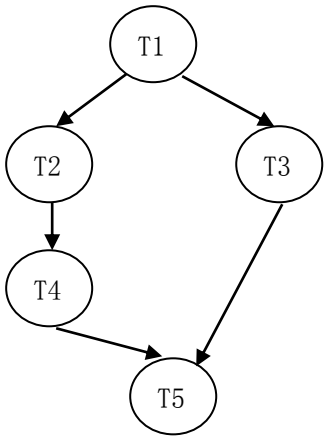

图 5.2 工作流 DAG 图

一个作业可以包含一个或多个 Task，以及表示其执行次序关系的工作流 workflow。工作流是一个 DAG 图（有向无环图），描述了 Job 中各个 Task 之间的

依赖关系和运行约束。简单的说，每个作业流对应一个 DAG 图，每个 Task 对应 DAG 图的一个节点。

云服务在这块的设计原则是尽可能简化用户配置，比如用户不需要自行设计这个 DAG 图，只需通过简单的 key:value 对的方式进行配置，通过配置中的 dependencies 来设置依赖关系即可。同时这个依赖关系必须是无环的，否则视为无效工作流。依赖配置表可简化如下：

表 5.1 作业流依赖关系配置表

作业名称	JobName
作业类型	Type
任务内容	XXX
依赖关系	Dependencies

用户指定了任务之间的先后依赖关系后，云服务系统会首先对这些任务构造 DAG 图，然后进行拓扑排序[30]。

排序输出算法步骤是：

(1) 初始化所有的任务节点入度为 0。

(2) 根据任务之间的依赖关系，构造点和点之间的有向边。如 Job A 依赖于 Job B，则构造 B 节点到 A 节点的一条有向边。

(3) 扫描所有节点，将入度为 0 的任务节点加入优先队列。

(4) 取出队列头节点，可以保存或执行该任务。遍历该节点的所有边，将其边的另外一个节点的入度减一，若减一之后入度为 0 则加入优先队列。

(5) 反复执行 4 步骤，直到队列为空。

(6) 检查所有节点的入度是否为 0，如果有一个节点入度不为 0，则判断依赖图有环，报警用户输入的作业流依赖关系无效。

数据迁移云服务系统关于作业流调度这块设计为三个部分，分 webserver,Executor server,database。如下图：

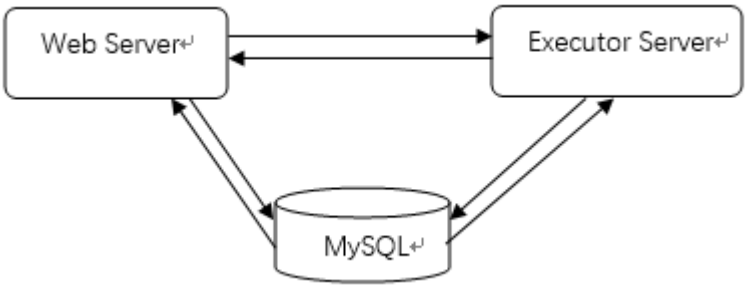

图 5.3 任务调度架构设计图

Webserver 主要是调度模块与用户交互的界面。用户可以通过该页面进行项目管理、作业流依赖配置、权限认证、任务执行状态查看等功能。Executor server 是一个集群，负责作业流、作业的具体执行及日志记录、状态反馈等。Mysql 数据库则用于存储作业流、作业执行状态、依赖关系等中间信息状态。

作业流执行过程概括如下：

（1）首先Webserver根据内存中缓存的各Executor的资源状态(Webserver有一个线程会遍历各个 active executor，去发送 http 请求获取其资源状态信息缓存到内存中)，按照选择策略（包括 executor 资源状态、最近执行流个数等）选择一个 executor 下发作业流。

（2） 然后 executor 判断是否设置作业粒度分配，如果未设置作业粒度分配，则在当前 executor 执行所有作业；

（3） 如果设置了作业粒度分配，则当前节点会成为作业分配的决策者，即分配节点；

（4） 分配节点从 zookeeper 获取各个 executor 的资源状态信息，然后根据策略选择一个 executor 分配作业；

（5） 被分配到作业的 executor 即成为执行节点，执行作业，然后更新数据库。包括任务执行的进度，状态等信息。

流程图如下：

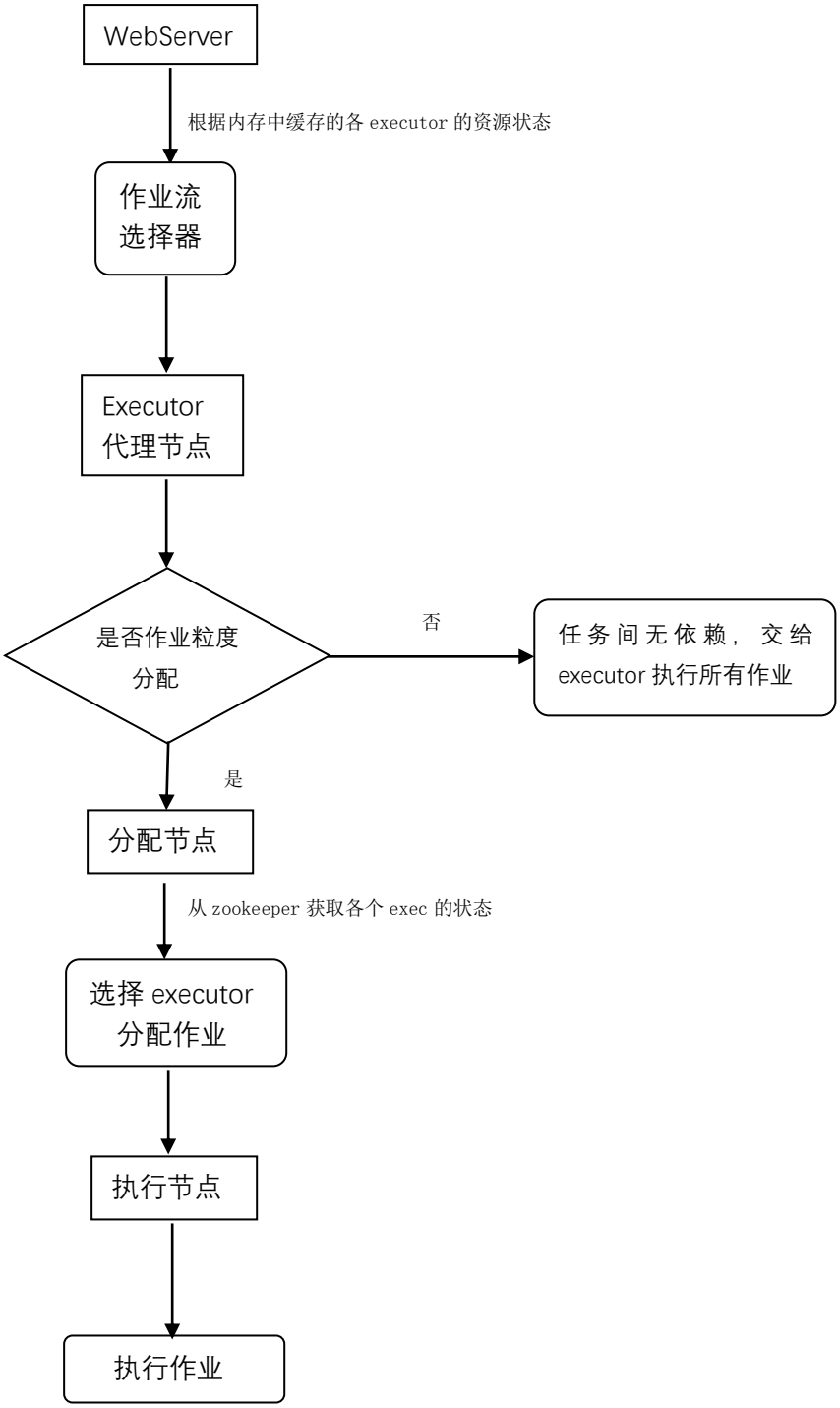

图 5.4 作业流管理流程图

5.4 监控报警模块设计与实现

5.4.1 监控数据收集与时间序列数据库存储

监控数据收集业内主要有两种模型，一种是 pull 模型，一种是 push 模型。

Pull 模型是建立在监控服务端，监控服务端一定时间周期性地从被监测节点获得、抓取相应监控数据。在监控服务端处理。是一种被动式地拉取信息方式。而 push 模型是建立在监测节点，也是在执行关键任务节点建立代理。代理主动把相应数据推送给监控服务后台。这种方式更为灵活，因为推送时间可以是周期性的，比如按照预先的配置，每隔几秒进行数据推送。同时也可以自适应地进行推送，比如被监控节点有数据变更时才进行推送。

本文中采用 Push 模型进行监控数据收集。Pull 模型有很大的弊端，比如周期性抓取数据，这会占用一定的带宽资源，而且周期时间并不好确定。而 push 模型更加智能。这里对于需要频繁收集数据的节点，会按某个周期推送监控数据，对于没有明确周期的节点可以根据相应数据变化作为触发条件进行监控数据推送。这样使得监控数据收集对于原任务执行节点的性能影响非常低。

监控数据的有效存储也非常关键，它是数据分析、报警预测、监控数据可视化的基础。监控数据的格式常常有时间戳这一字段。监控数据格式一般如下：

表 5.2 监控数据格式

timestamp	name	value	type
20161201	System.memory	60%	Physical server

时间戳字段代表了某一时刻的监控数据。监控数据往往是实时变化的，所以时间戳字段很关键。同时有监控项目名称，有 cpu、内存、IO 等指标。值域是个参数，如果超过一定的阈值，可能要采取报警会故障恢复等措施。类型是用户自定义字段，说明监控类型。有物理服务器和进程等等。对于不同类型的监控数据我们系统设计统一的存储模式，进行格式化统一存储。这个存储必须要快速，因为监控数据往往以流式数据大量涌入，传统的事务性关系数据库性能会存在问题。同时监控数据分析和预测也希望存储系统有很高的查询系能。

这里采用时间序列数据库[31]作为监控数据存储系统。基于时间序列的特点，关系型数据库无法满足对时间序列数据的有效存储于处理，而时间序列数据库为带有时间戳特性数据的读写做了很多优化工作，具有着很高的读写性能。而这一特性恰好能够满足监控数据存储读写需求。这里会采用开源的 openTSDB[32]数据库，它是用 hbase 存储所有的时序来构建一个分布式、可伸缩的时间序列数据库。它支持秒级数据采集所有 metrics，支持永久存储，可以做容量规划，并很容易

的接入现有的报警系统里。OpenTSDB 可以从大规模集群中获取相应的 metrics 进行存储、索引和服务，从而使得数据可视化更为简单和有效。

5.4.2 物理服务器节点监控

服务器物理主机监控[33]主要有：服务器节点增加、移除监控功能。即云服务系统增加了节点要能够及时被监控系统发现，注册并纳入监控体系；执行状态监测，如 CPU、内存等状态指标。

对于每个被检测节点，这里设计一个 monitor agent 代理服务进程，该轻量级进程负责与被检测节点交互。如新加入一个节点，monitor agent 会在 zookeeper 主节点进行注册。通常是在 zookeeper 主节点相应目录进行注册，比如约定好的/machines 目录存储了所有注册节点名称和 IP 等信息。/machines 目录下的文件记录信息，是监控服务端和 monitor agent 共享的数据读写区。当新节点加入，monitor agent 会把相应机器信息写入该目录；另一方面，监控服务端会周期性地扫描该目录，一旦发现新的目录，说明有新的机器节点加入了集群，此时及时把该节点纳入监控系统范围。

Monitor agent 还会周期性地探测服务器节点，作心跳检测。同时把心跳信号发往 zookeeper 主节点，zookeeper 会根据/machines 目录下节点名单进行比较，如果哪个节点返回来的心跳信号不正常或是信号丢失，及时反馈给监控服务后端服务，以作下一步处理。

对于服务节点执行状态的监控，monitor agent 会去使用被监控节点的系统调用获取相关数据。如 free -m 可以查看内存使用情况，top 命令可以查看进程 CPU 占用情况等。这些具体的检测数据会存储到 openTSDB 时序数据库作进一步分析，由于集群节点众多，收集的监控信息量也比较大。若多个节点监控信息直接写入时序数据库 opentsdb 是不合适的。因为这里的生产者消费者模型有这样的特点，生产者即监控数据产生端，监控数据产生速度没有一定的规律，且数据源很多。为了缓和数据读写，在监控数据产生端和数据消费者存储中间加上消息队列，这能够有效提高监控数据采集和读写效率。这里会采用 kafaka 做缓冲。监控数据从 kafaka 出来后还要做格式的归一化，最后用统一的格式保存在时序数据库 opentsdb 中。监控物理服务器的架构如下：

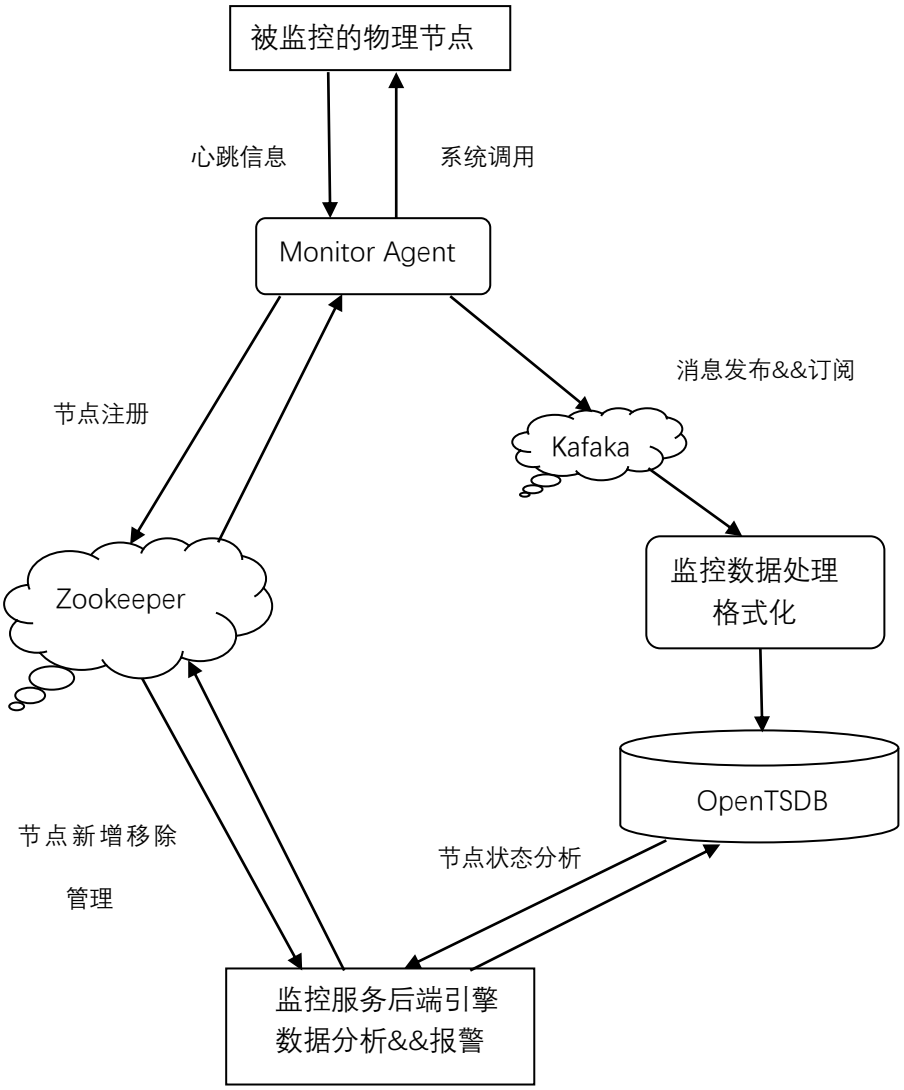

图 5.5 物理服务器监控架构

5.4.3 任务执行进程监控

任务执行进程监控[34]是监控系统的重要一个环节，直接关系到数据迁移云服务的核心迁移能否正常运行。数据迁移云服务中的重要进程有，数据源增量数据监视进程、关系型数据库日志同步进程、增量数据提取进程、数据传输进程等等。对于不同的进程监控策略略有不同，而这里希望能够针对进程监控的一般性设计相应的监控手段和策略。

进程监控要达到的目标如下：

- 能够对数据迁移云服务的关键进程进行监控和跟踪。云服务系统涉及的

节点众多、任务众多、进程众多。监控系统要能够对关键服务进程进行定位跟踪，能够周期性地获取进程状态。

- 故障快速定位和恢复。要尽可能保证数据迁移服务不中断。这里的云服务是面向众多用户、业务，数据迁移服务的稳定性不言而喻。作为公共服务要能够对用户的数据负责。面对大量数据任务，如果遇到了异常，或是执行任务进程死掉，要能快速定位和故障恢复。比如数据迁移任务执行了一半，遇到了数据源数据格式不正确导致的异常。监控系统能够及时发现，并进行重启进程和故障恢复。
- 进程监控要有利于云服务系统高可用性。分布式环境的复杂性，导致数据迁移过程进程死掉可能难以恢复。这时监控系统除了尝试重启进程外，也要能够进行任务节点切换。若原有节点无法再进行服务，要在其他备用节点生成相应进程继续进行服务。
- 对于重大故障的及时报警。这是进程监控的最后一步。即进程重启、任务节点切换都不能有效地恢复数据迁移任务。可能物理节点出现了永久磁盘损坏、系统断点等重大故障，监控系统要能及时通过邮件、短信报警。
- 监控数据的收集、存储、分析。监控数据虽然通常是一种实时的消费数据供任务调度等决策。但对于长期运行和维护的云服务来说。监控数据是重要的系统运行数据，反映了系统的性能和设计的优劣。监控数据的存储分析对于后来运维和开发人员查找系统设计缺陷、提升数据迁移云服务运行效率具有重大意义。因此，同物理服务节点监控一样，进程监控数据同样要存储在时间序列数据库 opentsdb 中。
- 进程监控数据可视化。这里与物理服务节点监控一致，通过 web 平台展示监控数据。

进程监控系统的架构与物理节点监控架构类似。监控资源信息表如下：

表 5.3 进程监控信息资源

监控名称	监控主题	编码类型
MonitorName	监控的进程名称	string
MonitorType	监控的进程类型(可自定义)	string
ProcessId	进程控号	long
FprocessId	父进程控制号	long
CprocessId	子进程控制号	long
startTime	进程开启时间	timestamp
runTime	进程已执行时间（ms）	long
status	进程执行状态(可自定义)	string
CpuPer	进程 CPU 使用量百分比	int
MemPer	进程内存占用百分比	int
IOPer	进程对写使用量百分比	int
NetDes	进程网络使用量描述	string
MonitorStatus	监控进程状态（数字定义）	int
extra	用户指定的监控项目	string

这里列出了进程监控的主要指标，同时支持用户扩展。

数据迁移进程监控逻辑如图：

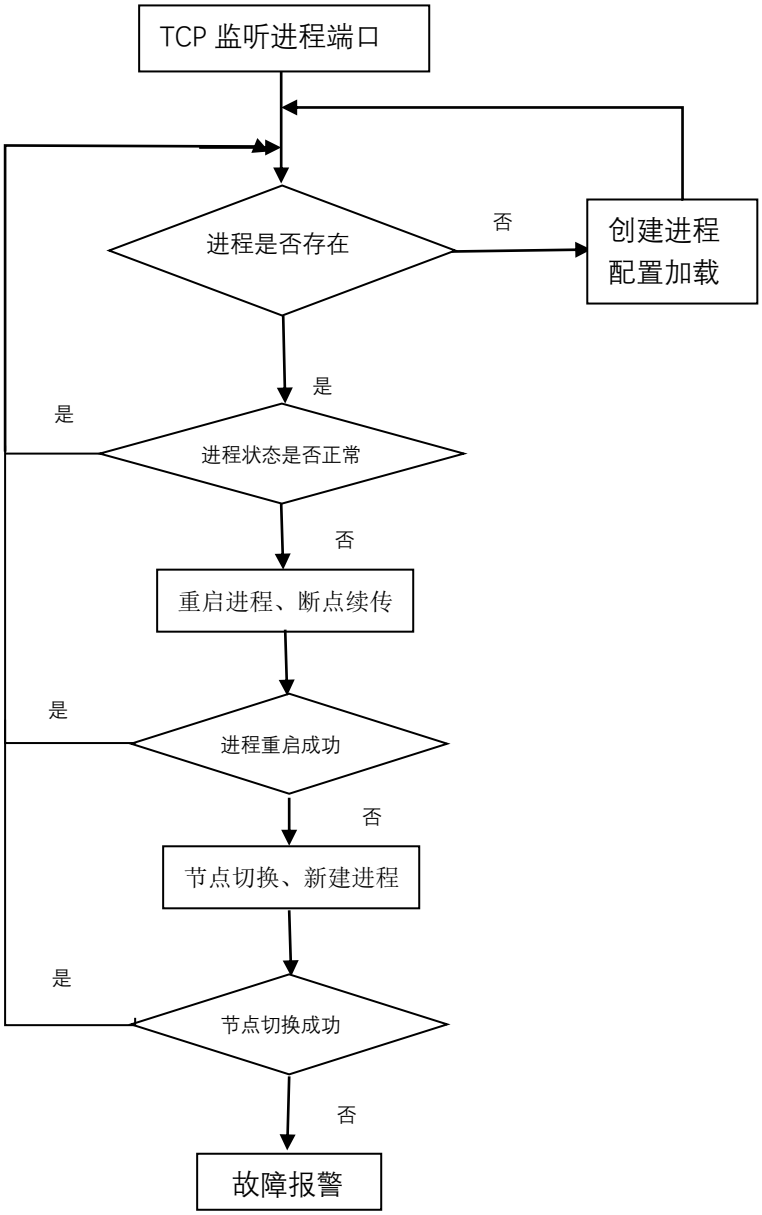

图 5.6 进程监控流程图

5.5 本章小结

本章论述了数据迁移云服务化关键模块设计与实现，使得数据迁移成为一种云服务、集群服务。集群管理主要介绍了主节点选举算法；基于因子分析和响应时间的负载均衡算法更为有效地评估了节点负载情况，防止了数据倾斜，这给数据迁移云服务带来了性能提升；作业流管理解决数据迁移任务间的依赖关系；监控报警模块有利于实现云服务的高可用性、健壮性、故障可恢复性。

第6章 实验与分析

本章主要是对数据迁移云服务核心模块功能、优化结果进行实验、分析、验证。主要考察流式数据迁移性能、负载均衡算法对于系统的改进等方面。

6.1 测试环境

这里是一个集群的服务，因此会涉及到多个节点。这里主要四个节点，分别为Node1，Node2，Node3,Node4。每个节点功能如下：

表 6.1 实验节点功能

节点名称	节点功能
Node1	Master 总控节点，部署 zookeeper
Node2	Binlog 同步存储节点
Node3	数据迁移执行服务节点，核心计算
Node4	数据迁移执行服务节点，核心计算

集群配置信息如下：

表 6.2 实验集群配置信息

项目名称	配置项	参数指标
执行服务集群	CPU	Intel Core i5-4460,3.20GHZ
	内存	8GB
	硬盘	250GB
	Mysql 版本	5.6.0。 端口：3306
	Mysql 信息	Row 级日志名称: Mysql_binlog_000001
	操作系统	Ubuntu 14.04
Hadoop 集群	Hadoop 版本	2.5.1
	Hive 版本	2.1.1
	HBase 版本	1.0.3

6.2 系统性能测试

本系统最主要的功能是数据迁移，分别为非结构化数据迁移和结构化数据迁

移，也是文件数据迁移和数据库 mysql 数据迁移。这里是迁移到 HDFS 集群。

6.2.1 非结构化数据迁移实验

对于非结构化文本文件。是有一个 agent 节点监听文件，一旦有内容变化，则创建迁移任务做数据传输。其网络拓扑图如下：

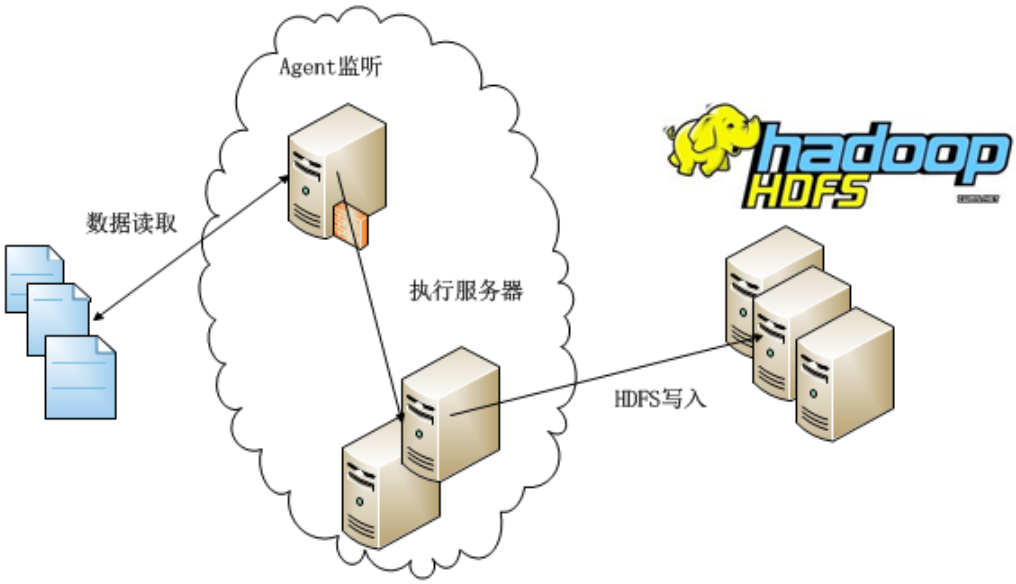

图 6.1 非结构化数据迁移拓扑图

对于非结构化数据，这里是首先生成若干 txt 文本文件。文件内容是一行一行的字符数字。文件内容有数据生成程序随机生成并写入。这里每隔一分钟，数据采集迁移一次，记录迁移的条数和耗费时间。一共收集 20 次数据。得到如下：

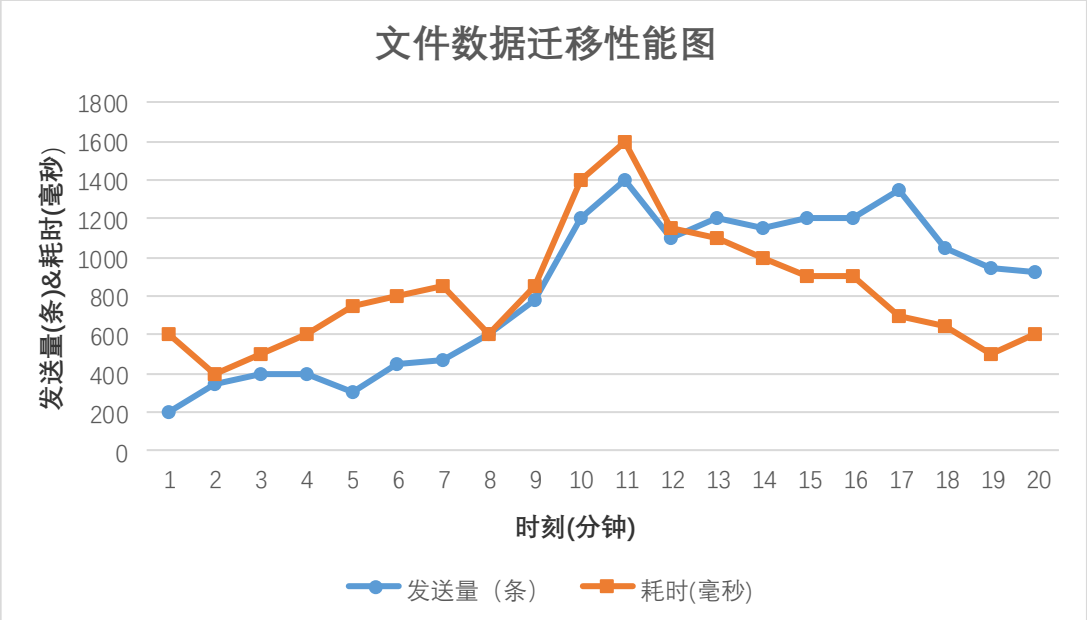

图 6.2 文件数据迁移性能图

由以上图分析，采集 20 次数据，生成内容随机。可以看到整体，生成的内容呈上升趋势，因为内容在实时产生，而迁移需要耗时且是周期性传输，因此会有数据积累。迁移耗时有一定的起伏，整体上，前半阶段，耗时高于传输条数，后半段耗时呈下降趋势。最开始耗时明显，因为最初执行服务节点要做迁移任务配置加载等初始化工作。之后一个阶段由于数据量增大，传输时间也同样增大，波动幅度不定也是集群工作与 HDFS 集群响应时间都在变化之中。一段时间后，任务耗时开始下降趋势，集群的负载均衡开始起作用，集群工作趋于稳定。

6.2.2 结构化数据迁移实验

对于结构化数据，这里主要是指关系数据库 mysql 数据。接下来，通过增加数据源个数，增加数据迁移 Job 个数，同时进行数据迁移，以测试系统并发吞吐能力。这里数据迁移任务数分别取 1,3,5,7,10,15 几种情况，测试迁移服务吞吐量。这里每个 mysql 数据源表格数据是事先生成好的。结果如下：

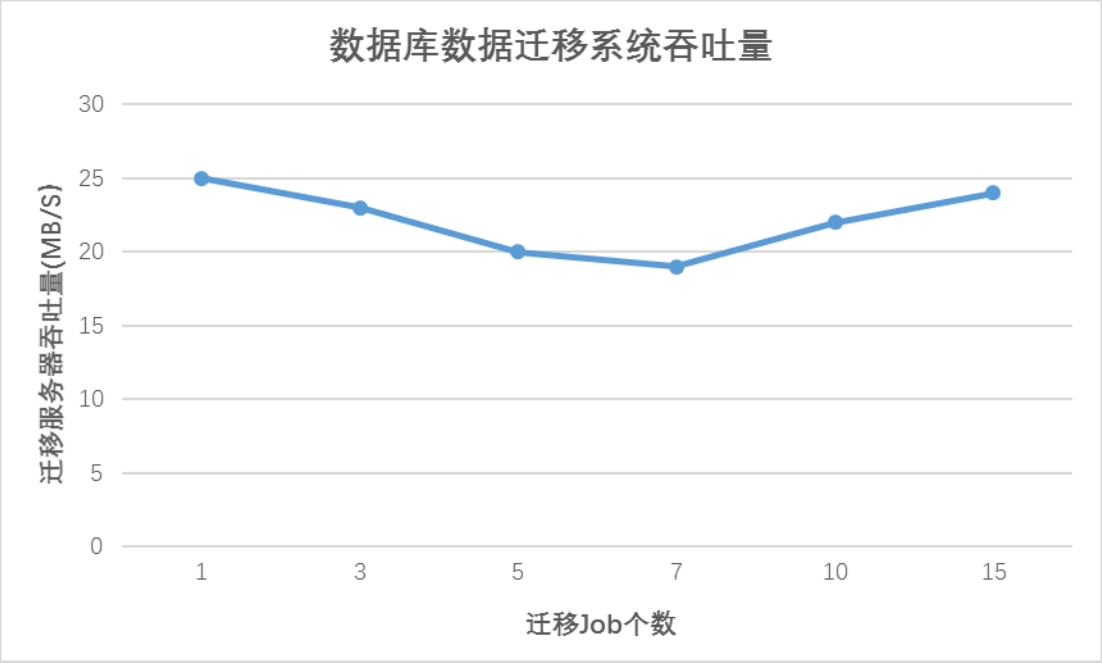

图 6.3 数据库数据迁移吞吐量图

由图分析得，随着文件个数的增多，系统吞吐量开始阶段略微有所下降而最终趋于稳定状态。即根据实验设备性能，系统吞吐量在 25M/S。系统吞吐量处于稳定，是因为这里执行服务节点有若干，他们是无状态平行模式，多个任务提交可以在多个节点上执行，这也是云计算的优势所在。

6.3 流式数据提取优化实验

流式数据提取，或是增量数据抽取，是流式数据迁移的关键问题之一。主要架构与非结构化数据迁移类似，会有 agent 节点监听数据源变化。而由于业务数据库数据的特殊性，即业务数据库往往处于繁忙状态，因此增量数据抽取往往不是直接在数据源上进行。由前文论述，业内已有工具的做法是将增量数据同步至备份节点，然后以备份节点作为新数据源进行迁移；而本文采用的是只备份数据库日志到备份节点，通过解析数据库日志，直接提取增量数据进行迁移。下面通过实验设计，对这两种方案进行性能比较。

实验方法是，事先在同一个数据库创建四张结构相同的表，然后分别插入 10,100,1000,10000 条数据。然后在增量数据抽取模块节点植入监控代码，即通过系统调用 system.currenttimemillis()去记录增量数据抽取时间，通过执行服务器性能监控脚本去查看服务器内存等占用率。

抽取时间对比实验数据如下：

表 6.3 流式数据抽取时间对比实验数据

流式数据量(记录数)	优化前提取时间(毫秒)	优化后提取时间(毫秒)
10	670	260
100	2980	1311
1000	15652	8402
10000	73516	29724

更加直观的折线对比图如下：

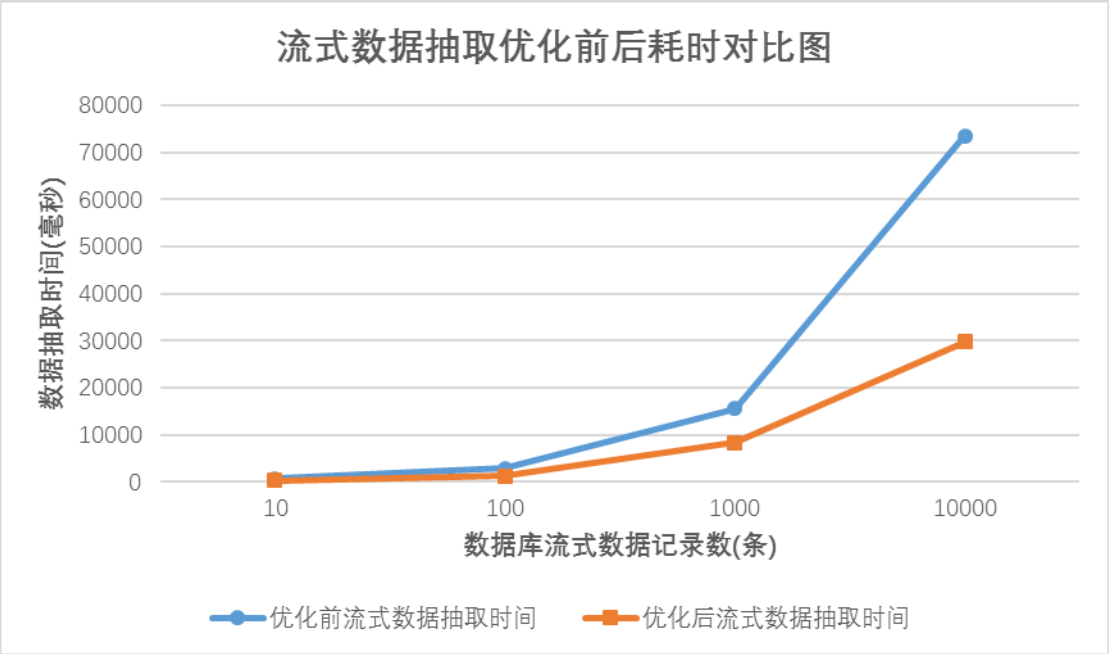

图 6.4 流式数据抽取优化前后耗时对比图

通过折线对比图，可以非常明显地看到优化前后的差异。当数据源数量小时，即数据源只有 10 条、100 条记录时，两种方案耗时很接近，并无大差别。因为此时由于数据量很小，无论是通过数据库备份方案，还是优化后的数据库日志数据提取方案，抽取速度都非常快。而当记录条数增加到 1000，尤其增加到 10000 条记录时，优化前后方案耗时差别明显。这是由于优化前方案，数据要写入新数据库，这个阶段由于要事务性写数据同时更新数据库索引，因此耗费了大量时间。而直接从日志文件提取增量数据，只有读文件时间和正则式解析文件时间，大大缩小了数据提取耗时。

这里也设计了内存消耗对比，实验数据如下：

表 6.4 内存消耗对比实验数据

流式数据量(记录数)	优化前内存占比	优化后内存占比
10	5.30 %	5.30 %
100	5.80 %	5.50 %
1000	8.30 %	6.40 %
10000	15.60 %	9.70 %

折线对比图如下：

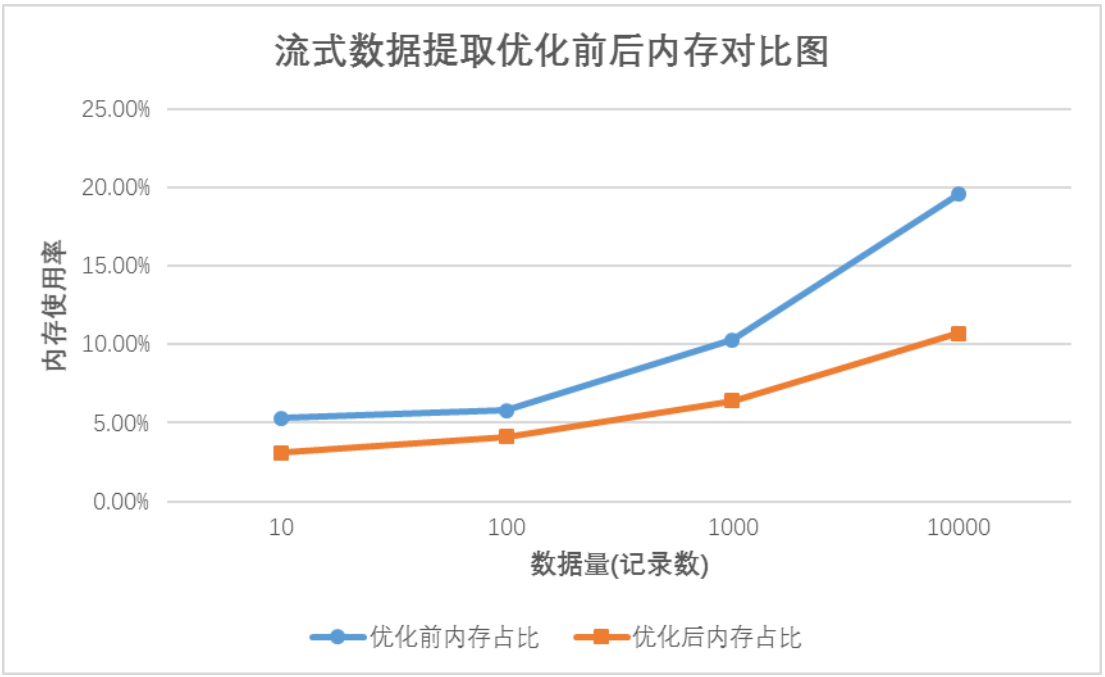

图 6.5 流式数据提取优化前后内存对比图

由上图可看到优化后内存占用率也降低了。可见数据库数据插入性能内存开销也不小。

6.4 负载均衡算法优化实验

由前文论述，本文提出了基于因子分析和响应时间的负载均衡算法 DLBRTFA。该算法在选择负载节点时，将数据迁移任务的执行时间考虑在内。认为如果节点正在执行的任务时间差别很大时，优先考虑任务执行时间少的。同时通过因子分析，得到动态评估节点负载的数学式。

本实验采用 OPNET Modeler 网络模拟仿真软件。通过实验，对比 DLBRTFA 与最小连接数算法 LCS 和经典动态负载均衡算法最小权值负载调度算法 WLL 进行性能比较。这里，实验将会考察请求响应时间。

实验网络拓扑图如下：

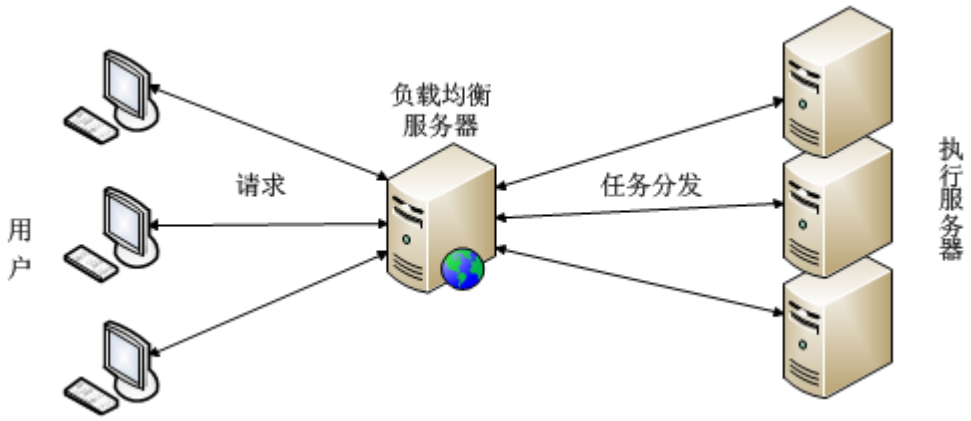

图 6.6 负载均衡网络拓扑图

实验配置如下：

表 6.5 负载均衡实验配置表

配置项	参数设置
服务器节点个数	10
服务器类型 A	权值最低(性能最低)
服务器类型 B	权值中等(性能中等)
服务器类型 C	权值最高(性能最高)
运行时间	20min
数据采集周期	30s
输入	模拟软件不间断发出请求
输出	请求响应平均时间

这里会针对 LCS、WLL、DLBRTFA 三种算法，应用于负载均衡任务分发中。分别考察算法对于 A、B、C 三种类型服务器的负载分发情况。以响应时间作为度量，考察三种算法对于负载数据的倾斜程度，节点负载状态判断情况。

实验结果如下：

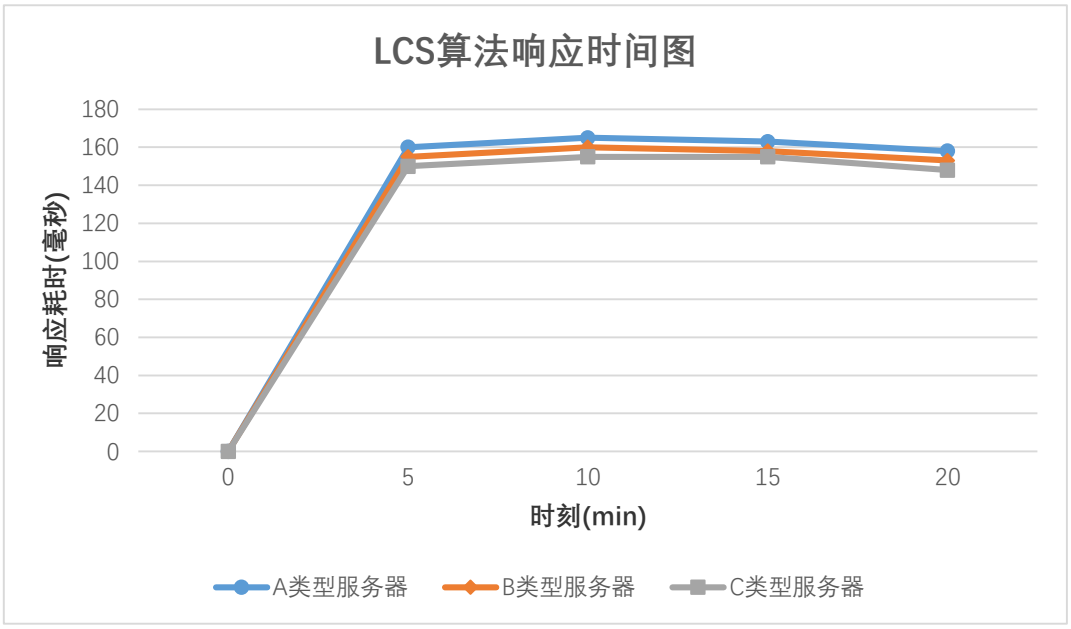

图 6.7 LCS 算法响应时间图

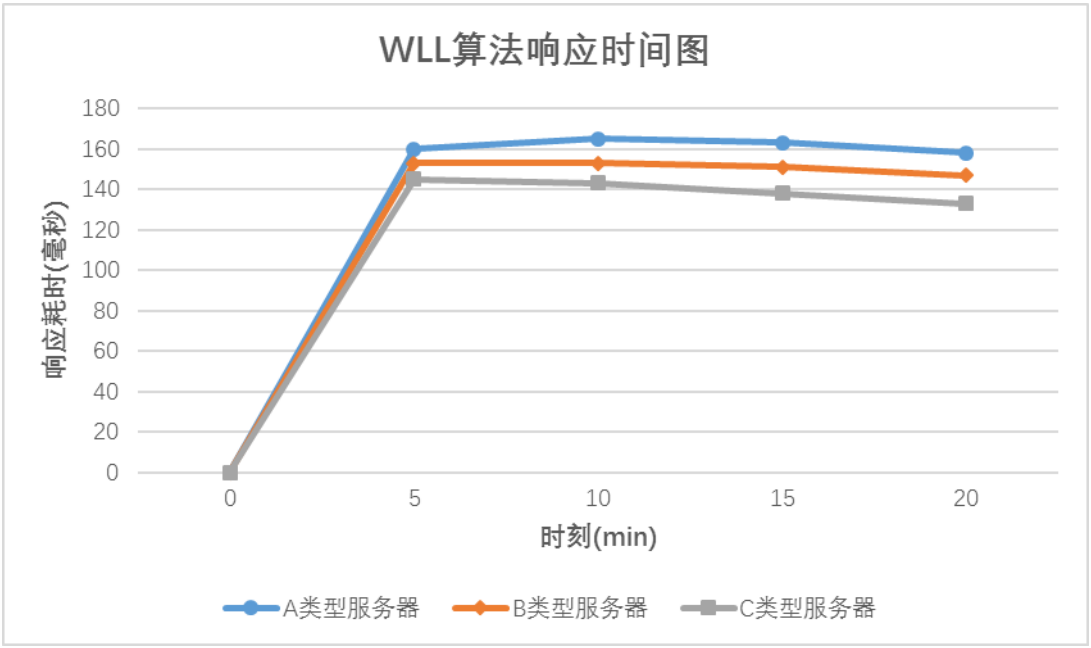

图 6.8 WLL 算法响应时间图

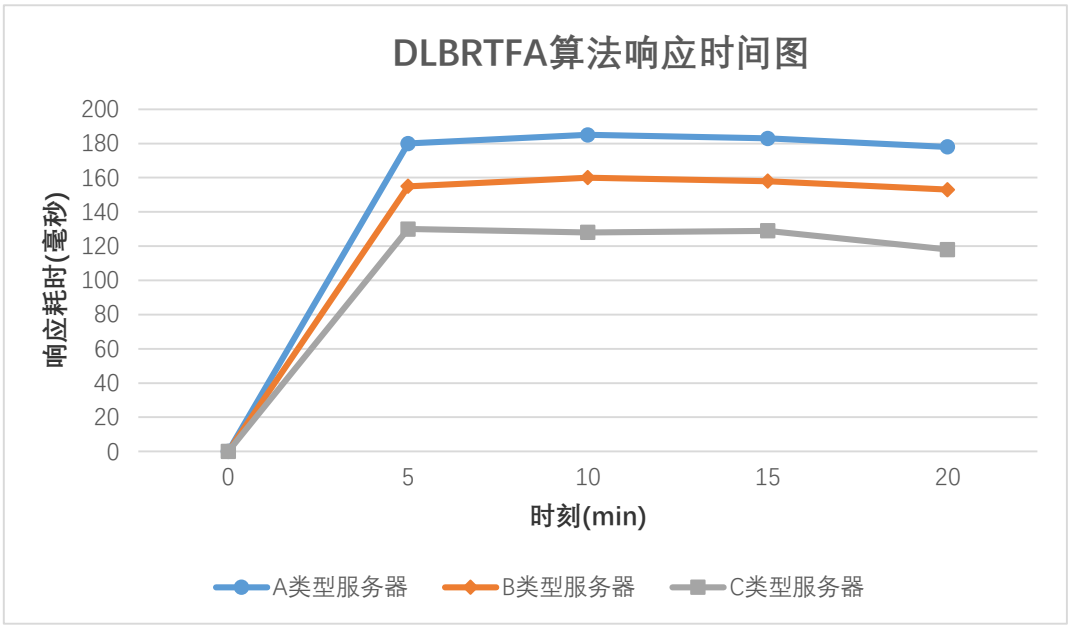

图 6.9 DLBRTFA 算法响应时间图

这里由于 A,B,C 三类服务器性能权值不一样。同样的请求，应该 A 的响应时间最长，C 的响应时间最少。理想的负载均衡算法应该能够明显区分三种类型服务器。由以上三张实验图可以看出。同样并发请求的情况下，对于同样的物理设备节点，三种算法对于三种类型物理设备的负载能力区分有所区别。LCS 是根据服务节点连接数或是任务排队数进行负载评估，连接少的任务其负载低。然而其并没有区分各种连接本身有一种权值在里头。比如有的连接耗时更高，有的则耗时低。图 6.7 中三种类型服务器响应时间接近，因此 LCS 负载均衡算法实际运行并不理想。对于 WLL 算法，其是一种动态负载均衡算法，其考虑了连接权值，在一定程度上有所改进,但是考虑的负载因素单一，实际的负载因素更多应当综合考虑。如图 6.8，三种类型服务器响应时间有所区分，但还不明显。而由图 6.9 可以看出,对于本文所提出的 DLBRTFA 算法,三种类型服务器响应时间区分明显。其代表了 DLBRTFA 算法使得集群资源得到了更好的利用,使得负载均衡服务器根据执行服务服务器动态负载情况更为科学地调度,这为数据迁移云服务整体吞吐量地提升势必会有很大的影响。DLBRTFA 算法有更好调度结果原因可总结如下：

(1) 常见的经典负载均衡算法如 LCS、WLC、WLL，这些算法考虑的负载因素

有限，如考虑连接数、请求数、节点物理性能。而实际的负载因素很多，如任务响应时间、CPU、内存、网络。且应该是动态变化的综合负载状态。本文论述的 DLBRTFA 算法考虑了常见的负载因素，且将他们综合起来考虑而不是单一考虑。

(2) 实时有效、客观的评估节点负载状态是负载均衡算法设计的难点之一，已有的算法往往根据人为经验调整负载。如 CPU 利用率或是内存占用率达到一定的经验值，就降低其负载。而本文论述的 DLBRTFA 算法尝试通过数学表达式去客观表示实际负载状况，通过因子分析和数学推导，将常见因素综合在数学表达式中，更为客观。通过实验也证明了算法的有效性。

(3) DLBRTFA 算法是一种动态调整的负载均衡算法。节点的负载状态在实时变化，而调度也应当是实时动态变化的。DLBRTFA 算法带有自适应调节，当某个节点负载情况大于集群平均负载情况，则认为其负载过高，此时会动态降低其负载；反之，则认为其负载偏低，可以分发更多数据迁移任务，让其资源得到更充分的利用。这体现了算法的动态调整功能、自适应分发功能。

6.5 本章小结

本章针对数据迁移云服务的核心功能，设计了具有针对性的实验。包括了数据迁移的性能实验，流式数据提取优化实验，以及负载均衡优化实验。论证了设计实现思想的科学性、有效性。

第7章 总结与展望

7.1 本文总结

本文基于已有的研究成果和工业界实践经验，从企业数据迁移的实际需求出发，设计了一个高可用、高性能、容错自恢复的流式数据迁移云服务。本文主要贡献如下：

（1）设计并优化了基于数据库日志的流式数据提取、迁移技术。通过对数据库日志进行解析，提取增量数据，并将这些数据直接封装为消息发往 hadoop 集群。大大降低流式数据提取的 IO、网络等开销。

（2）将因子分析数学思想应用于负载均衡负载状态评估，将响应时间纳入负载均衡参数指标。该算法相对于传统的负载均衡算法，能够更有效地评估节点当前负载情况，更大地利用好集群资源。大大提高了数据迁移系统的吞吐量和集群计算能力。

（3）将数据迁移系统上升到云计算的高度。针对业内已有迁移工具配置复杂、单机性能低下、容错性差等问题，本文提出的数据迁移云服务设计更够更好的提升系统整体迁移能力和吞吐。同时对于迁移任务具有一定的故障可恢复性。

7.2 下一步工作和展望

数据迁移是大数据时代一个重要的领域，数据分析处理的前提是已有数据向分布式平台的迁移。数据迁移也涉及到多方面的知识，如数据传输、网络、安全、分布式理论等。本文提出的设计思想是基于已有企业需求、有限的知识背景，对于海量数据迁移还没有测试，因此迁移性能、吞吐量还有待大数据的进一步测试与设计优化。同时数据迁移数据的完整性、安全加密也有待进一步考虑。

参考文献

[1] Bigdata Era[EB/OJ],http://www.csdn.net/article/2011-11-03/306923

[2] data transmission[EB/OJ],http://wiki.mbalib.com/wiki

[3] data integration and analysis[EB/OJ],http://wiki.mbalib.com/wiki/4320.index

[4] Sqoop Docs[EB/OJ],https://sqoop.apache.org

[5] Hadoop give the new need of data transmission [EB/OJ],http://www.infoq.com/cn/news/2014/11/hadoop-data-migration-tools/

[6] Web development[EB/OJ],http://developer.51cto.com/web

[7] Nosql technique[EB/OJ],http://nosql.org/21.index

[8] Hadoop Docs[EB/OJ],https://hadoop.apache.org

[9] mysql binlog[EB/OJ], http://dev.mysql.com/doc/

[10] InnoDB Storage Engine[EB/OJ],http://mysql.refman/8.0/en/innodb-storage.html

[11] 马宁.动态负载均衡算法的研究[J].华中科技大学学报:自然科学版,2010,38(2):23-35.

[12] 刘建，李绪志，一种动态负载均衡机制的研究与实现[J].计算机工程与应用,2006,27(4):142-145.

[13] Karger D,Sherman A,Berkheimer A,et al.Web WLC algorithms[J]. Computer Networks,1999,31(11):1203-1231.

[14] Keong Loh Peter Kok,Hsu Wen Jing,Wentong Cai.dynamic load balancing[J].IEEE Parallel and Distributed Technology, 1996,4(3):30-35.

[15] Sun ZhiWei.A cluster algorithm identifying the clustering structure[C].International Conference on Computer Science and Software Engineering,2008:47-48.

[16] zookeeper docs[EB/OJ], https://zookeeper.apache.org/

[17] 云计算核心技术解读[EB/OJ], http://www.open-open.com/lib/view/open1421131308984.html

[18] Jeffrey Dean,Sanjay Ghemawat.MapReduce:Simplified Data Processing on Large Clusters[J],OSDI.2008,51(1):110-112.

[19] 流式计算[EB/OJ],http://mt.sohu.com/20160305/n439484116.shtml

[20] D.Daniels,L.B.Doo,A.Downingetc.Oracle Symmetric Replication Technology and Implementation for Application Design[J].1995,23(2):467.

[21] 肖晓.基于 JMS 的数据同步技术研究与实现[D].湖南大学,2013.

[22] Apache.flume Docs[EB/OJ], https://flume.apache.org/documentation.html

[23] data flow model[EB/OJ], https://flume.apache.org/FlumeDeveloperGuide.html

[24] Flume.flume architecture[EB/OJ],https://flume.architecture.org/

[25] Flume.NG cloudera introduction[EB/OJ],http://jinnianshilongnian.iteye.com/blog/2261225

[26] 远程调用过程[EB/OJ], http://baike.baidu.com/

[27] Paxos 协议/算法[EB/OJ], http://www.cppblog.com/kevinlynx/archive/2014/10/15/208580.html

[28] 杨传辉.大规模分布式存储系统[M],59-60.

[29] 陈练达，曾国荪.基于因子分析的动态负载均衡算法[J].微型机与应用.2015(2):59-62.

[30] GiancarIo Fortino,Daniele Parisi,Vincenzo Pirrone,Giuseppe Di Fatta.BodyCloud:A SaaS approach for community Body Sensor Networks[J].Future Generation Computer System,2013,23:567.

[31] Klaus EIhardt, Rudolf Bayer.A Database Cache for high Performance and FastRestart in Database Systems[J].ACM Trans.Database Syst,1984,9:15-17.

[32] OpenTSDB.Opentsdb Docs.[EB/OJ]http://opentsdb.net/

[33] 孙健波.PaaS 云平台自动化部署和监控的设计与实现[D].浙江大学,2016

[34] Massie,M. The ganglia distributed monitoring system:design, Implementation and experience[J].Parallel Computing,2004,vol.30:23-24.

致谢

时光飞逝，研究生生活到了快要毕业的时候。读研期间，我收获了很多，包括学术研究、工程历练、做人做事等方面。

感谢陈刚老师。您对于学术研究方向把握、对于行业发展趋势的深刻解读让我受益匪浅。让我学会了结合自己兴趣、业内发展动态，如何去选择有价值的研究课题，如何做好课题，如何做好工程助力学术研究。我慢慢学会更加深入和理性地对待科研、工程问题。

感谢陈珂老师。读研期间，您多次对我进行指导和谈话。使得我更加积极乐观的对待生活。一直在您的照顾下，我得以开阔了眼界、锻炼了自身专业能力、个人良好修养。

感谢寿黎但老师。您严谨的学术态度、对工作的高要求严标准，使得我对于科研工作满怀崇敬之心。也在您的影响下，踏实工作，追求卓越。

感谢李环、赵王军、叶茂伟、顾晓玲、朱珠、骆歆远、王振华等师兄，你们对于科研和工作的热情，榜样的力量让我坚定了前行的脚步。感谢你们平时对我的细心指导。

感谢我的同学，金明健、朱华、钱宇、于志超、刘伟、冯杰等。和你们在一起的时光快乐而充实，学习和生活每一天都充满了朝气。有了你们，我也有了更加快乐的成长。

朱清华

2016-12-27