2017 届研究生硕士学位论文

分类号：_____ 学校代码：10269

密　级：_____ 学　　号：51141500029

華東師範大學

East China Normal University

硕士学位论文

MASTER'S　DISSERTATION

论文题目：可扩展数据库管理系统中的数据复制

院　　系：计算机科学与软件工程学院

专　　业：软件工程

研究方向：分布式数据库

指导教师：宫学庆　　教授

学位申请人：钱招明

2017 年 4 月 10 日

Dissertation for master's degree in 2017

University code:10269

Student ID:51141500029

East China Normal University

Title: DATA REPLICATION IN SCALABLE DBMS

Department:	**School of Computer Science and Software Engineering**
Major:	**Software Engineering**
Research direction:	**Distributed Database System**
Supervisor:	**Prof. GONG Xueqing**
Candidate:	**QIAN Zhaoming**

April，2017

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《可扩展数据库管理系统中的数据复制》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名：_____ **日期：** 年 月 日

华东师范大学学位论文著作权使用声明

《可扩展数据库管理系统中的数据复制》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文的著作权归本人所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和学校指定的相关机构送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

（ ）1.经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文*，于 年 月 日解密，解密后适用上述授权。

（ ）2.不保密，适用上述授权。

导师签名_____ 本人签名_____

年 月 日

* “涉密”学位论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

钱招明硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
张蓉	教授	华东师范大学	主席
钱卫宁	教授	华东师范大学	
翁楚良	教授	华东师范大学	
张召	副教授	华东师范大学	
沙朝锋	副教授	复旦大学	

摘 要

随着互联网的不断发展，数据规模不断增大，数据库系统的存储与计算的横向扩展能力将会越来越重要。因此，分布式数据库系统以其良好的扩展性受到了工业界和学术界的广泛关注。其中，基于日志结构存储（Log-Structured Storage）的分布式系统成为了一种新的趋势，这种读写分离的架构已应用于分布式数据库系统中，如阿里巴巴的开源关系型数据库管理系统 OceanBase。

数据导出是数据复制常见的技术之一，常用于企业级应用，来提高系统的可用性、可扩展性，以及保证数据的可靠性。在采用读写分离架构的分布式数据库系统中，由于数据分为静态数据和动态数据，并且静态数据存储于不同的物理节点上，数据复制成为了一种既消耗时间，也浪费系统资源的一种操作。本文主要分析了在读写分离的分布式数据库架构下，数据复制存在的问题，并提出了有效的解决方法。

本文工作的主要贡献如下：

1. 设计并实现了一种考虑负载均衡的静态数据导出方法。首先，针对分布式数据库的架构特点，直接向不同物理节点发起并发查询请求，减少数据的网络传输次数，缩短响应时间。其次，采用生产者消费者模型加快数据写磁盘速度并解决占用大量内存的问题。最后，根据数据多副本的特点，将查询请求均匀的发送给各个节点，使系统中的各个节点负载均衡，同时也能提高整体数据导出的性能。
2. 设计并实现了一种基于日志解析的动态数据捕获方法。一方面，实现日志同步和日志拉取功能，保证数据的正确性。另一方面，在日志解析过程中精简对同一元组的频繁操作，避免冗余操作，降低应用更新的代价。
3. 通过基准测试 YCSB 生成测试数据集并设计多组实验，验证了本文提出的数据导出方法的可行性与高效性。并在开源数据库 CEDAR 上实现了本文提出的数据导出方法。实验结果展示了本文提出的数据导出方法能有效的降低响应时间，减少系统资源占用。

本文提出的数据复制方法在 CEDAR 中的测试结果表明，该方法极大地提升了数据导出的效率。同时，本文提出的方法对同类型的可扩展数据库管理系统的数据复制有借鉴意义，也为可扩展数据库管理系统后续的数据复制技术提供了参考。

关键字：分布式数据库，基于日志结构存储，数据复制，数据导出，日志解析

Abstract

With the rapid development of Internet, the data volume becomes is expanding sharply. The scalability of storage and computing will become increasingly important in database systems. Therefore, distributed database system with good scalability has attracted the attention from industry and academia. An architecture based on log-structure storage (Storage-Structured Storage), which separates read and write, has become a new trend and been applied to distributed database systems, such as Alibaba's open source relational database management system (RDBMS) OceanBase.

Data export is one of the common technologies of data replication, which is used in enterprise applications to improve availability, scalability and reliability. In the distributed database system using read-write separation architecture, the data are divided into static data and dynamic data. Since the static data are stored on different physical nodes, data replication becomes an over-consumption operation. This paper mainly analyzes the existing problems of data replication under the distributed database architecture and puts forward an effective solution.

The main contributions of this work are as follows:

A static data export method for load balancing is designed and implemented. First, owing to directly accessing to the physical storage nodes, our method avoids the limitations of a single middleware merge node. Second, the producer-consumer model takes full advantage of the disk throughput and avoids excessive consumption of memory. Finally, a data export strategy based on multi-copy data is proposed, which further improves the performance of data export.

A dynamic data capture method based on log analysis is designed and implemented. On the one hand, we ensure the correctness of log synchronization and log extraction. On the other hand, we leverage log compaction to reduce the log volume, which also reduces the cost of applying of writes.

Through the benchmark YCSB, we conduct multiple experiments to verify our proposed methods. We implement our approaches in the open source database system CEDAR. The experimental results show that the data export methods have a good performance in terms of

response time and can reduce the system resource consumption effectively.

The testing results of data replication method proposed by this paper in CEDAR demonstrate that method greatly improves the efficiency of data export. In the same time, the method proposed by this paper not only equips with reference meaning for same type of scalable database management systems, but also provides the reference for the subsequent data replication technology of scalable database management system.

Key Words：*Distributed Database，Log-structured Storage，Data Replication，Data Export，Log Parser*

目 录

插 图

第一章 绪论

1.1 研究背景

作为计算机领域的一个重要分支，数据库技术一直倍受学术界和工业界的关注。从上世纪 60 年代末开始，传统数据库经历了三个阶段的发展，从层次数据库[1]到网状数据库[2]再到关系数据库[3]，其中关系数据库应用最为广泛、影响最为深远。

关系模型的概念是在 1970 年由 Edgar Codd 提出的[4]，奠定了关系模型的理论基础。关系模型有着严格的数学基础，较高的抽象级别，且容易理解和使用。关系模型用关系来表示实体和实体之间的联系，即用二维表的形式表示数据的逻辑结构。此后，关系数据库迅速发展，市场上主流的数据库几乎都是基于关系模型的，例如甲骨文公司的 Oracle 数据库[5]、IBM 公司的 DB2[6]和微软公司的 SQL Server[7]等。

随着计算机和网络技术的普及和快速发展，数据存储技术得到了很大的进步，同时也面临着巨大的挑战[8]：用户数量的不断增加，数据分布的地域空间更加广阔和数据量爆发式的增长，这些对数据库系统的可扩展性、高性能、高可用等方面提出了更高的要求。为了应对这些挑战，各种新型的分布式数据库[9]便应运而生，如 Google 公司的 Spanner[10]、Apache 的开源项目 Hbase[11]和阿里巴巴公司的 OceanBase[12]等。

分布式数据库因具有良好的可扩展性而受到工业界的广泛关注。数据的分布式存储是其主要的特点之一，通常分布式数据库按照一定的策略将数据水平切分或垂直切分成许多数据分片，并且每个数据分片都有多个数据副本，这些数据副本存储在系统中的不同服务器上。由于多副本的存在大大提高了系统的可用性，通常系统中会有一个主控节点，负责系统中数据存储节点的负载均衡[13,14]。此外，系统中的数据存储节点是无状态的，因此可以随着数据量的增长，动态的向

系统中添加服务器，以满足海量数据存储的需求。

随着互联网的快速发展，新应用、新模式不断涌现，传统应用逐渐被“互联网+”应用所替代[15]。应用迭代速度的加快促使企业业务系统频繁更新升级。同时，数据规模的增长和应用需求的变化，包括系统升级、数据备份和 7*24 小时服务，推动着数据库系统的升级与替换。因此，不同的业务系统间需要大量的数据共享和迁移[16]，这就需要从一个数据库系统中将数据导出，然后再将数据导入到另一个数据库系统中。但是，数据导入导出会对数据库系统造成很大的压力，如何做到高效快速的完成数据复制并且不影响数据库系统的性能，这成为运维和管理中的重要问题[17]。

传统的关系数据库都有各自的数据复制方案。数据复制通过冗余数据提高数据库管理系统的可靠性与可用性，使本地数据读取成为可能，节省了远程数据访问的网络通信时间，缩短了查询响应时间。此外，当节点出现故障或网络异常时，数据复制技术能保证系统正常对外提供服务[18,19]。传统的数据库管理系统，单节点集中式存储，复制简单。但对于可扩展数据库，如何充分利用其分布式存储，多机多 CPU 及节点间高速网络等特点，这是值得研究的。

此外，数据复制技术还可用于分离 OLTP 和 OLAP[20,21]。在传统的数据复制技术中，由复制代理通过网络传输数据快照文件或日志文件完成出版和订阅数据。目标数据库经复制代理订阅源数据库的最新数据版本，在本地执行加载数据操作或回放日志操作，可实现源数据库与目标数据库的最终一致性。目标数据库高度自治，能很好的支持 OLAP 应用，提高了查询处理的性能，也缓解了源数据库的压力。因此，研究数据复制是很有必要的。

数据导出是数据复制常见的技术之一，常用于企业级应用，来提高系统的可用性、可扩展性，以及保证数据的可靠性。本文通过分析分布式数据库 CEDAR 的架构和底层存储特点，提出了两种数据导出方法。

1.2 本文贡献

基于开源的分布式数据库 CEDAR（基于 OceanBase0.4.2 版本研发的）设计

并实现了一种考虑负载均衡的并行静态数据导出方法。此外，还设计并实现了一种基于日志解析的动态数据捕获方法。贡献如下：

1. 设计并实现了一种考虑负载均衡的并行静态数据导出方法。首先，通过导出服务器直接向 CEDAR 的基线数据服务器发起静态数据导出请求，减少了静态数据从基线数据服务器到合并服务器的网络传输时间。其次，通过在导出服务器端并行处理子请求结果集，缩短了请求的响应时间并解决了占用大量内存的问题。最后，根据数据分片的分布信息将请求均匀的发送给各个基线数据服务器，充分利用集群中服务器的资源且避免单点负载过高，同时也提升了性能。
2. 设计并实现了一种基于日志解析的动态数据捕获方法。一方面，在导出服务器内部实现日志拉取和日志同步功能，保证了持续获取动态数据的正确性。另一方面，在日志解析过程中对同一元组的频繁操作进行精简处理，避免了对同一元组的冗余操作，降低了应用更新的代价。
3. 通过基准测试 YCSB 生成测试数据集并设计多组实验，验证了本文提出的数据导出方法的可行性与高效性。我们在开源数据库 CEDAR 上实现了本文提出的数据导出方法。实验结果展示了本文提出的数据复制方法能有效的降低响应时间，减少系统资源占用。

本文提出的两种数据导出方法在国内某大型商业银行的基于分布式数据库 CEDAR 的历史库系统中得到了实际应用。历史库系统主要用于存储和管理海量历史交易数据，并且支持查询和分析用户的历史交易信息。

1.3 本文结构

依据本文的主要研究工作，本文结构安排如下：

第二章，首先，简单介绍分布式数据库 CEDAR 的实现架构和基本内容。其次，对本文的研究问题进行描述并分析其难点。最后，介绍本文工作的相关技术和概念。

第三章，针对分布式数据库 CEDAR 设计并实现了面向静态数据的数据导出

方法。接着，提出两种基于不同粒度的负载均衡算法。

第四章，针对分布式数据库 CEDAR 设计并实现了两种面向动态数据的数据导出方法，介绍不同的应用场景。

第五章，首先，介绍实验环境、实验对比方法和基准测试。然后，通过分析实验结果，与传统的数据复制方法进行对比，说明本文的贡献。

第六章，是对全文工作的总结。

第二章 问题描述与相关工作

2.1 CEDAR 介绍

分布式数据库 CEDAR 是基于 OceanBase0.4.2 研发的可扩展关系数据库，可支持数百 TB 的海量数据，数十万 TPS 和数百万 QPS 的访问量，并且支持跨行跨表事务。CEDAR 数据库基于基线数据和增量数据分离的存储架构，即基线数据与增量数据分别存储在系统中不同的节点上，而且该数据库采用了单点写入的模式。基线数据是只读的，所有的修改操作都更新到增量数据中，系统会定期将更新服务器上的增量数据融合到基线数据服务器上的基线数据中，这个过程称为每日合并。

2.1.1 架构介绍

CEDAR 系统中包含四个重要的模块，分别是：主控服务器 RootServer、更新服务器 UpdateServer、基线数据服务器 ChunkServer 和合并服务器 MergeServer，其整体架构如图 2-1 所示[24]。

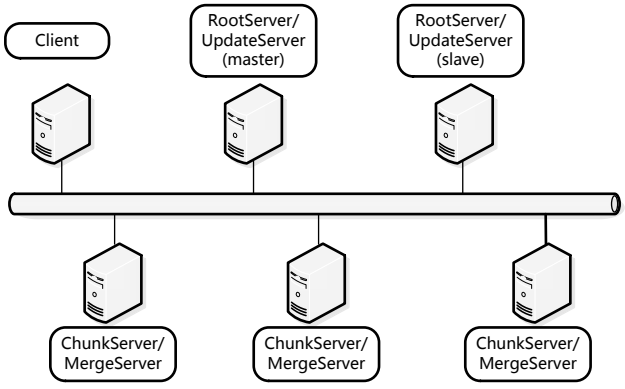

图 2-1 CEDAR 整体架构图

1. 主控服务器 RootServer：简称为 RS，管理集群中所有的服务器，是整个集群中的总控节点，记录数据分布情况和管理集群中的副本。RS 通过租

约机制选择唯一的主 UpdateServer。此外，RS 定期会与 MergeServer、ChunkServer 通信，从而感知 MergeServer 和 ChunkServer 是否在线。

2. 基线数据服务器 ChunkServer:，简称为 CS，用于存储 CEDAR 系统的基线数据，通常基线数据会有多个副本存储在不同的节点上。CEDAR 数据库将基线数据切分成多个大致相等的数据分片，称为 Tablet。集群中的 RS 会根据每个 CS 上的 Tablet 数量进行负载均衡。
3. 更新服务器 UpdateServer：简称为 UPS，存储 CEDAR 系统所有的增量更新数据，是系统中唯一接受写入操作的节点。为了保证可靠性，UPS 在执行更新操作之前需要首先写操作日志，然后将更新操作写入内存数据结构中，称为内存表 Memtable。
4. 合并服务器 MergeServer：简称为 MS，接收客户端读写请求的接口，客户端和合并服务器间采用了原生的 MySQL 通信协议。MS 负责接收并解析 SQL 请求，然后将请求转发给对应的 CS 或 UPS，如果请求涉及多个 CS，则 MS 会将多个 CS 返回的结果集进行合并生成最终结果，并返回给客户端。

2.1.2 底层存储

数据分布： CEDAR 中的数据是由静态数据和增量数据两部分组成的。静态数据分布在多个 CS 上，而增量数据存储在 UPS 上，增量数据也称为动态数据。整个集群中只有一个 UPS。如图 2-2 所示，在 CEDAR 集群中有 5 个 Tablet，每个 Tablet 有 3 个数据副本，它们分别分布在 4 个 CS 上。在 RS 上有一个类似于有序表格的数据结构，称为 RootTable，它负责维护每个 Tablet 所在 CS 的信息。如图 2-2 所示，在 RootTable 中记录了 Tablet(1)的三个副本分别存储在 CS1，CS3 和 CS4 上。UPS 存储了所有 Tablet 对应的增量更新数据，最新版本的数据存储在内存表中，内存表实质上是一颗高性能的内存 B+树[25]。

基线数据划分：在 CEDAR 数据库内部通过主键对表的数据进行排序，主键可由一列或者多列组成，且唯一。排序后的数据被水平切分成数据量大致相等的范围，称为 Tablet。例如，一张表的主键范围是[1，300]，它被划分成了三个 Tablet。

则划分后的 Tablet 的主键范围分别是[1，100]、[100，200]和[200，300]。通常，每个 Tablet 的默认大小为 256M（可配置）。当前，CEDAR 数据库采用一级索引结构 RootTable，该结构存储在主控服务器 RS 上。RS 能够检测到集群中的发生故障的 CS，会触发对这台 CS 的 Tablet 副本增加操作。此外，RS 也会定期执行负载均衡操作，从高负载的 CS 中迁移某些 Tablet 到低负载的 CS 中。

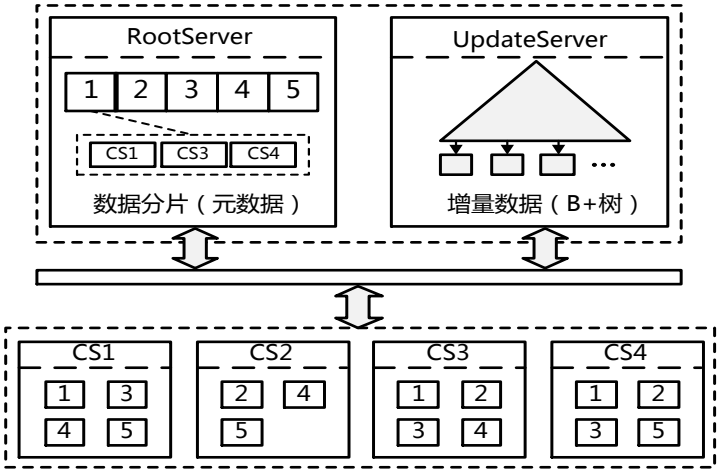

图 2-2 CEDAR 数据分布图

2.1.3 传统的数据导出与导入

数据复制最简单的方式就是从源数据库中将数据导出到数据文件中，然后再利用目标数据库的加载工具进行导入。目前，CEDAR 可用的数据复制方法是使用 DataX[26]导入/导出数据，其结构模式如图 2-3 所示。DataX 是阿里巴巴集团研发的一个可实现异构数据库之间的数据交换的工具，其工作流程大致就是用 Reader 模块从源数据库读取数据，通过 Storage 模块将 Reader 模块读到的数据交换给 Writer 模块，Writer 模块将数据写入目标数据库。其中，DataX 通过 JDBC 的方式读取源数据库的数据，即使用 SQL 的方式。由上文可知，CEDAR 中的数据分为基线数据和增量数据两部分。因此，CEDAR 的数据导出可以分为全量数据导出（基线数据+增量数据）和基线数据导出。

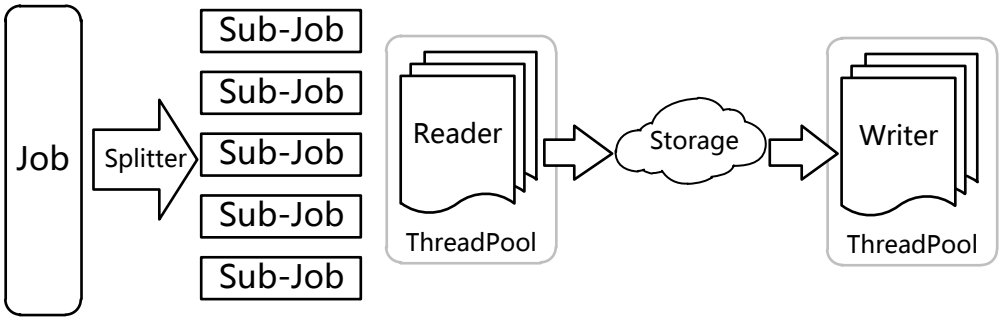

图 2-3 DataX 结构模式图

全量数据查询：在 CEDAR 中，SQL 语句的执行流程如图 2-4 所示。首先，合并服务器 MS 负责接收 SQL 请求，MS 对 SQL 语句进行词法解析和语法解析，生成 SQL 语句的逻辑执行计划和物理执行计划。其次，MS 根据本地缓存的 Tablet 信息，会把请求发送给所有涉及到的 Tablet 所在的基线数据服务器 CS。第三，CS 会请求更新服务器 UPS 获取该 Tablet 所对应的增量更新数据，CS 将增量更新数据与本地的基线数据融合后的结果返回给 MS。第四，当 MS 接收到 SQL 请求涉及到的所有 Tablet 数据时，将所有结果合并后返回给客户端。

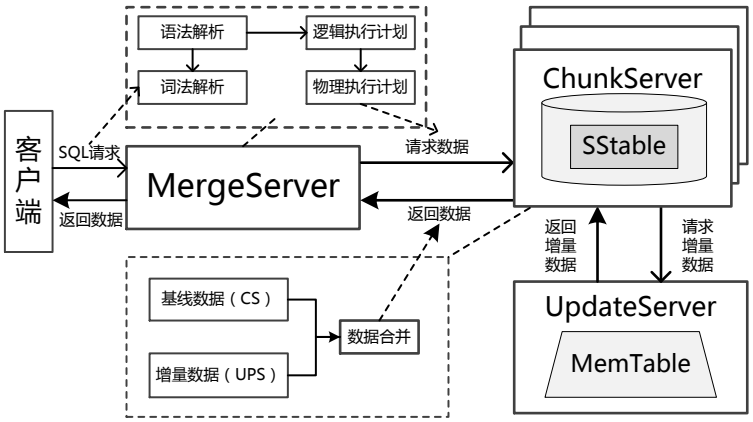

图 2-4 CEDAR 查询执行流程

基线数据查询：在 CEDAR 中，查询基线数据和全量数据的 SQL 语句仅有细微的差别。CEDAR 支持在 SQL 语句中加入 HINT 命令/*+read_static*/，当 MS 接收到 SQL 时会解析其中的 HINT 命令，生成与全量导出 SQL 不同的逻辑执行计划和物理执行计划。如图 2-4 所示，当 CS 接收到 MS 发来的请求后，不会再向 UPS 获取 Tablet 对应的增量数据，而是直接将 Tablet 数据返回给 MS。

2.2 问题描述

CEDAR 中的数据复制是指在保证数据一致性的前提下将数据库中的数据抽取出来并通过高速网络连接转储到远端节点的非易失性存储内，如图 2-5 所示。在下文中没有特别说明的情况下，数据导出就是指数据复制。在实际应用中有多种方式实现数据复制，既可以通过 SQL 查询的方式，又可以采用转发事务日志的方式。但是在实际应用中，数据复制不应该影响 CEDAR 本身的查询与事务处理性能。

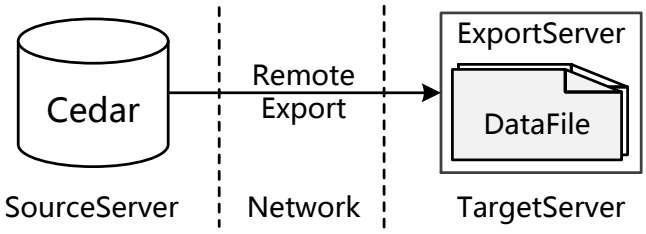

图 2-5 CEDAR 中的数据复制示意图

通常，当数据库版本升级、应用迁移和运行环境发生变更时，需要用到数据复制技术。此外，数据复制也可看作是备份技术的一种。对于少量数据的复制，可以直接采用 SQL 查询的方式将数据导出到指定的结构中。但是，对于海量数据的复制，应考虑数据库的查询处理能力。

虽然数据库复制已经有很多实现方法，但是 CEDAR 的数据复制面临着一系列特殊的问题。首先，作为一个分布式数据库，CEDAR 中的数据量是巨大的，如果采用传统的单节点单线程的 SQL 查询方式将非常耗时。其次，所有的查询结果都存储在查询节点的内存中会造成内存溢出。第三，同样重要的是，在复制期间 CEDAR 仍在对外提供服务，需要考虑数据一致性的问题。第四，数据复制会占用系统较多资源，会影响其性能。

但是，从另一方面考虑，CEDAR 作为分布式数据库提供了多节点（多 CPU 多核）、并行访问，且节点之间高速网络。这些硬件资源为 CEDAR 的数据复制提供了很好的硬件保证。此外，CEDAR 具有静态数据与动态数据分离存储的特点，这也为我们实现针对 CEDAR 的数据复制提供了更多的思路。

因此，一方面，本文研究了基于分布式存储的静态数据的管理机制。在该机制下，提出了基于微分片的负载均衡的数据导出方法。该方法在保证了数据一致性的前提下，达到了快速高效地完成数据导出的目的。另一方面，本文研究了基于读写分离架构的动态数据的管理机制。在该机制下，提出了基于日志解析的动态数据捕获方法。该方法在保证了正确性的前提下，达到了持续获取动态数据的目的。

2.3 难点与挑战分析

如何将海量数据在指定的时间内完成分布式数据库 CEDAR 中的数据导出是本文需要解决的问题。通过研究分析 CEDAR 的架构，总结出影响其数据导出的几点因素，下面详细介绍：

1. 效率低。采用传统的 SQL 查询方式从 CEDAR 中导出海量数据不仅效率低下，而且会对数据库的查询节点造成非常大的压力。CEDAR 中的数据分布存储在多个节点上，当 CEDAR 处理查询任务时，需要将所有涉及到的数据从多个存储节点中拉取到查询节点上。查询节点需要对所有存储节点返回的结果做拼接合并处理，随着数据规模的增长，查询节点的处理时间越长。
2. 内存不足。CEDAR 在处理查询任务时，所有的结果都保存在查询节点的内存中。当查询结果超过其内存大小时，查询节点会将结果写入磁盘。待查询节点收到所有结果后，重新从磁盘中读取数据并响应查询请求。一方面，读写磁盘的操作耗时严重，可能造成查询请求超时。另一方面，海量数据的查询任务对数据库造成巨大的压力，影响了其它应用程序对数据库的访问性能。
3. 数据一致性。数据库是重要的基础服务，通常需要保证 7*24 小时可用。在数据导出期间，CEDAR 仍在对外提供服务，若是不对数据导出做相应的控制，则最终导出的数据不是同一个时间点上的快照数据。例如，数据库中有两个表，分别是 A 表和 B 表。在 T1 时间点，开始导出 A 表的

数据，耗时 t1。在(T1+t1)时间点，开始导出 B 表的数据，耗时 t2。若是在 T1 到(T1+t1)时间段内，B 表的数据发生了变化，则导出的数据是不一致的。

2.4 相关工作

2.4.1 数据复制

数据复制是指在两个或者多个数据库系统之间拷贝数据的过程。在这个过程中，将主数据库中的数据拷贝到从数据库中，以支持分布式应用。数据复制将中心数据库的数据分发到网络中的其他节点，保证各个节点的数据一致性[27]且每个节点高度自治，用户可以通过访问本地节点替代访问远端节点。因此，数据复制技术能很好的支持分布式应用，也可用于在开发阶段模拟真实生产环境服务于测试工作。

数据复制包括全量复制和增量复制两种类型。例如，把源数据库中表 A 的全部数据复制到目标数据库中的表 B 中，这个过程称为全量数据复制。假设在某一时刻 T1，表 A 的数据与表 B 的数据完全相同，然后在一段时间间隔 t 内对表 A 执行过增删改等操作，则在 T2 时刻（T2=T1+t），把表 A 在 t 时间内的数据变化复制到表 B 中，保证两表的数据一致，这个过程称为增量数据复制。

2.4.2 数据复制常用方法

基于触发器法[28,29]：在源数据库中需要复制的表中建立相应的触发器，每当对表中的数据进行更新操作时，如插入、更新和删除，并且更新操作成功写入磁盘时，此时触发器就会被激活，将数据的变化序列写入临时表中，再由相应的程序定时将临时表的内容传递给目标数据库，保证目标数据库与源数据库的数据一致性。这种基于触发器的数据复制方法，不仅要在源数据库上进行大量的编写触发器的工作，而且它会占用数据库相当多的资源，会对数据库的性能有较严重的影响。此外，这种方法对网络的可靠性要求比较高，一旦网络出现故障，就无法正常工作。基于触发器的数据复制方法不仅实现复杂，且对源数据库要做较大

的改动，不容易管理，只能实现单向数据复制。因此这种数据复制方法不适用于企业级的需求，仅可满足较小的数据库应用。

基于日志法[30,31]：通常数据库系统都采用 WAL（Write-Ahead Logging）[32]技术来保证事务的 ACID 特性[33]，即在对数据库中的数据进行修改之前会将对数据的具体操作写入到日志文件中，则数据库的日志文件中记录了完整的数据变化序列。因此，我们可以通过提取和解析日志文件来获数据的变化序列，并且在快照数据的基础上进行更新操作的回放，使目标数据库与源数据库保持数据一致。这种方法的实现的大体思想是，在源数据库和目标数据库两端分别部署复制代理，在源数据库上的复制代理会定期将数据库的日志文件通过网络发送给目标数据库，然后目标数据库上的复制代理接收到日志文件后，将其解析成相应的 SQL 语句，并在目标数据库上重新执行这些 SQL，从而达到与源数据库的数据一致。这种方法占用源数据库很少的资源，并且适用场景比较广泛，能满足任何类型的数据复制需求，对数据的完整性有一定保障，并且在很小的延迟内就可以追赶上源数据库上的数据状态。但使用这个方法的前提是，能获取源数据库的日志文件，并且知道其日志的内部格式。不同数据库系统的日志文件格式不完全相同，不具有通用性。

基于时间戳法[34,35]：基于时间戳的数据复制方法是根据数据上一次的更新时间来判断此记录是否为增量数据，若为增量数据则可根据这条记录对快照数据进行相应的修改。实现这个方法的主要思想是，为应用系统中的所有表增加一个时间戳字段，每次对表中的数据进行修改操作时，记录下此次数据的修改时间。若想获取增量数据，每次对数据进行扫描筛选出修改时间戳大于快照数据中的时间戳即可。这种方法的不足之处是需要对应用系统做较大的改动，虽然对原有应用系统的运行性能几乎不会有什么影响，但是还存在对某些数据变化的捕获困难，即那些数据的变化并非由应用系统引起。虽然可以采用触发器来记录数据的变化时间，但同时又引入了新的问题，使系统的整体性能下降。这种方法使用条件较为严苛，存在明显的局限性。

2.4.3 数据复制研究现状

当前主流的关系数据库，如 Oracle、SQL Server、Sybase 和 MySQL 等都有各自的数据复制方法[22,23]。随着应用需求的不断变化，它们在已有的数据复制功能基础上增加了许多的功能组件，提供更为强大的功能[36]。

Oracle 中的复制技术：Oracle 数据库自身有一项集成的功能专门用于数据复制，这个功能组件名是 Oracle Replication[37]。Oracle 的数据复制技术本质是调用 Oracle Replication API 来实现的。Oracle 提供了两种复制方案：基本复制和高级复制。

1. 基本复制：通过在本地服务器节点（目标数据库）定义一个与远端服务器节点（源数据库）数据对应的快照数据，这个快照数据可以是数据库中的全部表或者是数据库中的部分表，然后源数据库定期将本地的增量数据传输给目标数据库，使两个数据库在某个时间点上达到数据一致的状态，在目标数据库中只能对数据进行读取操作，因此基本复制又被称为单向复制[38]。
2. 高级复制：通过在源数据库和目标数据库上设置触发器，当任一数据库对本地的数据进行增删改等更新操作时，触发器就会被激活，将数据的变化序列实时或者一定延迟后传输给远端数据库。在高级复制中，源数据库与目标数据库是对等关系，都可以进行查询和更新操作，并采用触发器的方式将对数据的修改操作发送到远端节点完成数据同步，保持节点之间的数据一致性，因此高级复制又称为对称复制[39,40]。对等复制与单向复制的对比见图 2-6。

单向复制	对等复制
仅主数据副本支持写入	多个数据副本均可读写
不会出现数据冲突	会出现数据冲突
数据结构可以不一致	数据结构一致

图 2-6 单向复制和对等复制的对比

SQL Server 中的复制技术：SQL Server 的复制是一系列技术的集合，包括

从存储转发到数据同步再到数据的一致性维护。在 SQL Server 的复制方案中主要分为三个角色：发布服务器、分发服务器和订阅服务器。发布服务器负责发布源数据库的内容，分发服务器负责转储发布服务器发布的内容，订阅服务器主动向分发服务器订阅其需要的内容，其中发布服务器和分发服务器可以是同一个节点。SQL Server 提供了三种复制方案：快照复制、事务复制和合并复制[41]。

1. 快照复制：发布服务器通过快照代理将所有要发布的所有表生成一份快照数据，然后通过分发服务器一次性分发给订阅服务器。而最近发生更新的数据不会被要求发布。
2. 事务复制：订阅服务器首先从发布服务器经过分发服务器获取一份完整的快照数据。发布服务器会把事务日志中记录的数据变化序列传输给分发服务器，分发服务器对数据变化序列进行排序，并在一段时间后将其分发给订阅服务器或者在订阅服务器主动请求时立刻发送变化序列，订阅服务器在快照数据上应用这些变化序列。要求发布服务器与订阅服务器保持稳定的网络连接。
3. 合并复制：由合并代理完成初始化工作，将快照文件由发布服务器同步到订阅服务器上，在发布服务器和订阅服务器断开连接时允许对各自节点的数据进行更新，当它们重新连接时，有合并代理完成交换自上次同步以来更新过的数据。两个服务器可能对同一数据都进行了更新，则在合并数据时可能产生冲突，遇到冲突并不是以发布服务器优先处理，而需要按照合并代理设定的冲突解决规则进行相应处理。

Sybase 中的复制技术：Sybase 的复制方案是基于日志的，有专门复制组件 Replication Server[42]。通过在源数据库上部署一个组件 LTM（log Transfer Manager），它实质是一个运行在源数据库端的进程，负责监控并捕获数据库上的事务日志，每当数据发生变化时，它就会将事务日志进行解析并将相应的操作序列发送给复制服务器，在复制服务器中有一个稳定队列，可以缓存由 LTM 发来的消息，复制服务器可实时的或在一定延迟后同步目标数据库[43]。

Sybase 的复制方案可支持同构数据库的复制，也可支持异构数据库的复制，但源数据库必须是 Sybase。对异构数据库的复制过程当中，不仅要在目标数据库

部署复制服务器，还在为目标数据库提供专门的 LTM 组件。Sybase 的复制方案实现简单，最多只是为非 Sybase 数据库构造 LTM 而已，它的架构虽然有较好的开放性，但只支持单向复制，且对网络的要求较高。

MySQL 中的复制技术：MySQL 的复制方案是基于日志的，可实现主从模式的数据库复制[44,45]。MySQL 实现复制方案的基本原理是主数据库会把本节点发生的更新操作写入一个日志文件中，称作二进制日志。日志内容大部分是更新数据的 SQL 语句。从数据库从远端节点获取日志文件，并将日志解析成 SQL 语句在从数据库上执行。

从数据库上主要有两个工作线程，分别数 I/O thread 和 SQL thread。其中，I/O thread 负责连接主数据库并请求二进制日志，接着 I/O thread 将接收到的二进制日志转储到本地的临时日志文件中；SQL thread 负责读取临时日志文件，并将其解析成解析成 SQL 语句，在从数据库上重新执行。MySQL 的数据复制方案增加了系统的健壮性，且对主数据库的营销较小，但其只支持数据库级别的复制，且不支持异构数据库复制。

第三章 面向静态数据的导出

3.1 概述

传统的数据导出方法是通过接口从源数据库抽取数据，例如 JDBC 或专用数据库接口。从 CEDAR 数据库中导出静态数据，首先通过接口向集群中的合并服务器 MS 发起静态数据查询请求。接着，MS 到请求涉及到所有基线数据服务器 CS 去请求数据。最后，MS 在内存中将所有 CS 返回的数据合并生成最终结果后返回。通常，这种方法只适用于少量数据导出。而对于海量数据，在数据导出过程中可能存在一些问题，具体如下：

内存瓶颈：上文提到 MS 需要合并所有 CS 返回的结果集，在合并操作过程中所有的结果都是存放在内存中的。当数据量大到超过 MS 的内存时，会引入磁盘 IO 的时间开销。另外，MS 合并结果集操作的时间开销也会随着数据量的增长而增长。因此，通过查询请求的方式从 CEDAR 中导出海量数据，不仅会占用 MS 的大量内存资源和 CPU 计算资源，而且会影响其他应用程序对 CEDAR 的访问性能。

网络传输：如图 2-4 所示，在 CEDAR 数据库中实现静态数据导出，数据需要经过 2 次网络传输，从 CS 到 MS，再由 MS 将合并后的结果集后返回。当 MS 接收到 SQL 请求时，会根据本地缓存的 Tablet 分布信息，将 SQL 子请求发送给相应的 CS。接收到子请求的 CS 会调用自身的 CS-SQL 模块做相应的计算，并将计算结果返回给 MS。接着，MS 合并所有 CS 返回的结果，得到完整的结果后返回客户端。

数据一致性：在数据导出期间系统仍在对外提供服务，本章提出的面向静态数据的数据导出方法避免了更新操作对的数据一致性的影响。但是，CEDAR 本身存在每日合并操作，将更新服务器 UPS 上的增量更新数据发送到 CS 上与其本地的静态数据融合生成新版本的静态数据。若是在数据导出期间，CEDAR 发

起每日合并操作并且比数据导出先完成，则导出的数据有一部分是旧版本的静态数据，另一部分是新版本的静态数据。

针对以上问题，我们设计并实现了一种基于 Tablet 分片的并行数据导出方法。在 CEDAR 中，表的数据按照主键排序后划分为多个大致相等的范围，称为 Tablet。根据这个特点，我们将表的数据导出任务拆分成多个 Tablet 的数据导出任务。通过导出服务器 ExportServer 与 CS 直接交互，跨过 MS 减少了 1 次数据网络传输的时间。ES 以 Tablet 为单位向集群中的多个 CS 同时发起数据导出请求，只要 CS 返回数据，ES 就可以立刻对数据进行处理，无需再做所有子结果集的合并操作，节省了一部分时间，同时也避免了占用大量内存的情况。ES 向 CEDAR 发起静态数据导出请求后，其大体执行流程如图 3-1 所示。

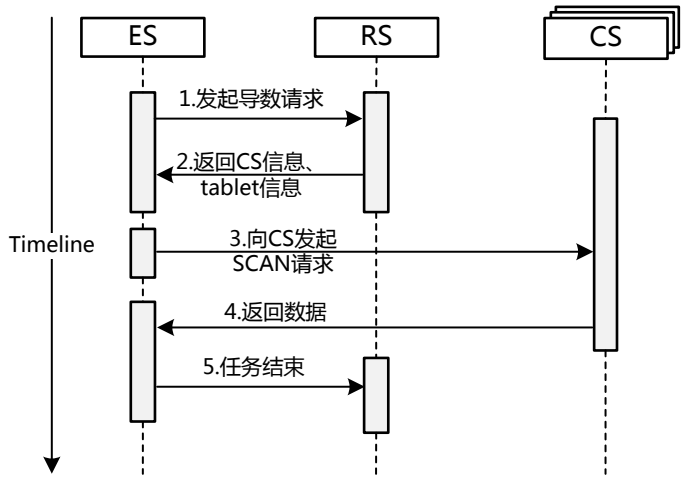

图 3-1 基于 Tablet 的数据导出执行流程

3.2 基于 Tablet 的数据导出

在 CEDAR 中采用传统的查询请求的方式完成数据导出，不仅效率低下，而且会对 MS 造成很大的压力，占用其大量的内存资源和 CPU 计算资源。此外，在数据导出期间还会影响其他应用程序对 CEDAR 的访问性能，甚至可能会造成 MS 对外部所有的请求返回响应超时。因此，我们设计并实现了一种基于 Tablet 的数据导出方法，通过 ES 直接向 CS 发起数据导出请求，不仅减少了数据网络传输时间，而且不会占用 MS 的资源。

3.2.1 基本思想

CEDAR 通常会按照主键对表的数据进行排序，并将排序后的结果划分为多个数据量大致相等的范围 Tablet，并且 Tablet 存储在多个基线服务器 CS 上。首先，由导出服务器 ES 向 CEDAR 集群中的主控节点 RS 请求读取 RootTable 获取 Tablet 信息。然后，根据 Tablet 的分布信息和主键信息，ES 构造多个 SCAN 请求，并将其发送至对应的 CS。

图 3-1 展示了基于 Tablet 的数据导出方法的大致执行流程。该方法主要由三部分组成：ES、RS 和 CS。其中，ES 是核心组件，负责发起和控制整个数据导出过程。整个执行过程主要分为如下五个步骤，下面介绍每一步骤的具体操作。

步骤一：ES 向 CEDAR 集群中的主控节点 RS 发起数据导出请求，该请求中附带多个参数，包括表名和保留当前静态数据版本的标记 tablet_version_retain。当 RS 接收到 ES 发送的请求时，构造 Response 信息，具体的信息如下。

1. 添加集群中所有可用的 CS 信息。
2. 添加当前静态数据的版本号，tablet_version。
3. 添加 RootTablet 中相应表的 Tablet 信息，包括位置信息(svr_ip, svr_port)、数据版本号 tablet_version、起始主键和结束主键，还有 Tablet 大小及数据记录数等。
4. 设置 tablet_version_retain 标记。如果在数据导出期间，CEDAR 集群正好发起每日合并操作，RS 会根据 tablet_version_retain 标记的值，通知 CS 是否保留旧版本的静态数据。

步骤二：RS 将构造好的 Response 信息发回给 ES。当接收到 Response 信息时，ES 进行如下处理：

1. 读取 CS 信息，对每个 CS 构建连接线程池，1 个 CS 可对应多个连接线程。
2. 初始化数据缓存队列，用于接收 CS 返回的数据包
3. 初始化数据结构 Tablet_Cache，用于存储 Tablet 信息。
4. 读取 Tablet 信息，用位置信息(svr_ip，svr_port)、数据版本号 tablet_version、

起始主键 start_rowkey 和结束主键 end_rowkey 来构造数据结构 TabletInfo 并存入 Tablet_Cache 中。

5. 初始化数据文件，文件以“表名_起始主键_结束主键.txt”命名。
6. 格式转化模块读取配置文件进行初始化。
7. 初始化 IO 线程并加载数据转化模块。
8. 不断从 Tablet_Cache 取出 TabletInfo，用表名、起始主键、结束主键和静态数据版本号多个参数构造 SCAN 请求，并根据 TabletInfo 中的位置信息将 SCAN 请求交给相应的连接线程处理。

步骤三：ES 中的连接线程将 SCAN 发送给对应 CS。接收到 SCAN 请求后，CS 调用 ObAgentScan 类的 scan 函数接口从本地磁盘读取 Tablet 数据，然后将数据封装成数据包并进行序列化操作。

步骤四：CS 将 Tablet 的数据包逐个返回给 ES。当接收到数据包时，ES 进行如下处理：

1. 初始化数据结构 TabletBlock，然后将数据包的内存块拷贝到 TabletBlock 中。
2. 将 TabletBlock 放入该 CS 对应的数据缓存队列中。
3. IO 线程监测缓存队列是否为空。首先，取出 TabletBlock 并把它反序列化成数据记录。接着，调用格式转换模块对每一条记录的相应字段进行处理。最后，将格式化后的数据记录写入对应的数据文件中。

步骤五：当 ES 接收到所有 CS 返回的数据时，会向 RS 发送心跳信息，通知 RS 数据导出任务已经结束。RS 接收到 ES 的消息后，会重置 tablet_version_retain 标记。

该方法通过 ES 直接向 CS 发起数据导出请求，数据直接传输到 ES 上而不需要再经过 MS，不仅减少了一次数据网络传输，而且不会占用 MS 的资源。此外，我们将表的数据导出任务转化成可并发执行的 Tablet 数据导出任务。ES 将 CS 返回的数据包直接放入数据缓存队列，然后由单独的 IO 线程对数据包进行处理。ES 不需要对所有的 Tablet 数据进行合并，而是并发的将 Tablet 数据写入文件。一方面，省去了合并操作的时间。另一方面，不会占用大量的内存。

3.2.2 实现细节

实现该方法的关键在于 ES，其中主要的功能模块如图 3-2 所示，下面会详细介绍各个功能模块。

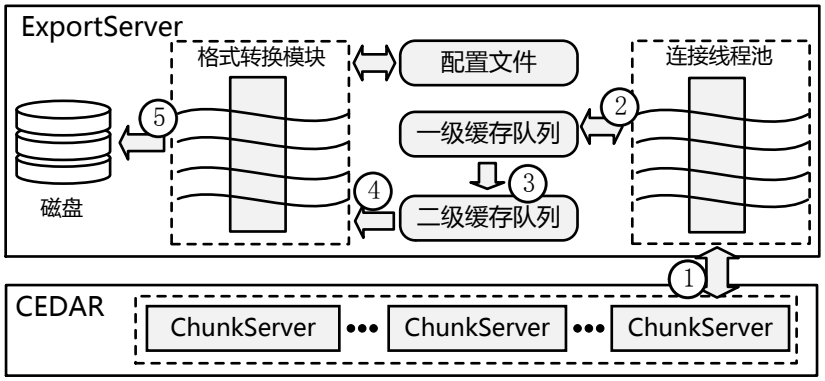

图 3-2 ExportServer 内部各功能模块

连接线程池：根据 RS 返回的集群中可用 CS 信息，为每个 CS 创建多个连接线程（可配置）放入线程池中。等待 SCAN 请求的调用，发送给对应的 CS。当 SCAN 请求得到响应时，将 CS 返回的数据包的内存块拷贝到 TabletBlock 中。接着，将 TabletBlock 放入数据一级缓存队列。数据包中会带有一个标记，表示是否还有后续的数据包。

数据缓存队列：由两级缓存队列组成，采用数据结构 list<TabletBlock>实现并采用生产者-消费者模型。其中，一级缓存队列服务于生产者，二级缓存队列服务于消费者。为了不产生冲突，给每个缓存队列分配一把锁（假设为 L1 和 L2）。生产者或消费者要操作缓存区之前，必须先拥有相应的互斥锁。消费者将二级缓存队列读完后，尝试获取 L1。若是消费者能拥有 L1，则会把一级缓存队列的内容全部拷贝到二级缓存队里中，然后释放 L1。

配置文件：在配置文件中可指定行分隔符、列分隔符。此外，还为字符型字段加双引号以及指定时间类型的字段导出到文件中的格式如：YYYY-MM-DD，YYYYMMDD 或 YYYY-MM-DD HH:MM:SS。

格式化模块：读取配置文件中的参数，来初始化 DataFormator 数据结构。本模块内有多个 IO 线程。IO 线程从二级缓存队列中取出数据后需要调用数据格式

化模块的 format 函数对数据做格式化处理。

IO 线程：每个数据缓存队列对应多个 IO 线程（可配置）。当数据缓存队列不为空时，IO 线程首先取出数据包并将其反序列化成数据记录。接着，IO 线程调用格式转化模块，对数据记录的相应字段进行处理。数据是以纯文本形式存储，默认采用 CSV（Character Separated Values）格式来存储每一条记录。

3.2.3 分析与讨论

本文提出的基于 Tablet 的数据导出方法相比于传统的基于 SQL 的数据导出方法从以下三个方面进行了改进：

1. 网络传输：本章提出的方法通过模拟 MS 向集群中的 CS 直接发起 Tablet 的数据查询请求，减少了一次网络传输。需要特别注意的是，我们减少的是数据从 MS 到客户端的网络传输，其中 MS 是以单线程的方式向客户端返回数据的。
2. 内存占用：本章提出的方法一方面不需要对 CS 返回的结果做合并操作，另一方面采用生产者消费者模型，多个 IO 线程不断的从数据缓存队列中取数据并发的将数据写入文件中。因此，本章的方法既减少了大量内存的占用，又提升了数据导出的速度。
3. 数据一致性：面向静态数据的导出方法虽然解决了由更新操作带来的数据不一致问题，但是 CEDAR 本身的每日合并操作会引起静态数据版本的变更。我们对 CEDAR 的每日合并流程进行简单改造，使其在数据导出期间保留旧版本的静态数据。我们在数据导出请求中加入当前静态数据的版本，这就解决了数据一致性的问题。

但是，本章提出的方法还有一些不足之处。CEDAR 中的数据通常会有多个副本存储在集群中的不同 CS 上，而 ES 默认向 Tablet 第一副本所在的 CS 发送 SCAN 请求，未考虑到集群中 CS 的负载均衡。若是大多数 Tablet 的 SCAN 请求都发送到了一个或者少数几个 CS 上，不仅没能充分利用集群中 CS 的资源，有可能会造成某些 CS 的单点负载过高而影响数据库对其他应用程序的服务能力。

3.3 负载均衡的优化策略

在 CEDAR 集群中数据通常有三个副本，以 Tablet 的形式分布在集群中的不同 CS 上，并且多个副本可以同时对外提供服务。如图 3-3 所示，Tablet1 有三个副本，它们分别存储在 CS1、CS3 和 CS4 上。RS 中的 RootTable 是基于序数组实现的，Tabet 的三个副本信息顺序地记录在 RootTable 中。MS 处理查询请求时，会根据本地缓存的 Tablet 分布信息，将请求转发给相应的 CS。

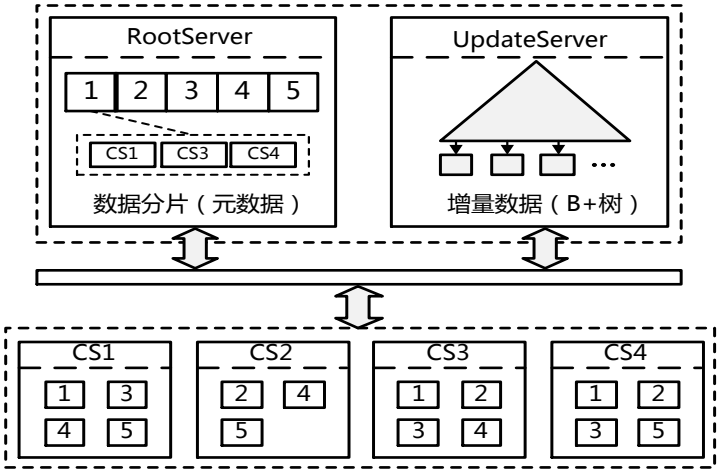

图 3-3 数据分布图

若是 Tablet 的数据读取请求都集中在少数几个 CS 上，则会导致这些 CS 的负载很高，而其它 CS 的负载很低。这样不仅没有充分地利用所有 CS 的资源，而且还可能成为影响数据导出性能的一个因素。

我们考虑如何充分利用 CEDAR 的数据有多个副本的特点，使 CEDAR 集群中各个 CS 上的数据读取请求负载大致相同。如图 3-3，若 Tablet(1, 3, 4, 5)的数据读取请求都被发送到 CS1 上，则会导致 CS1 比其他服务器的负载高出很多，没有利用好其他 CS 的资源。比较理想的情况应该是将多个 Tablet 的数据读取请求均匀的发送到集群中不同的 CS 上。例如，在图 3-3 中，Tablet(1, 4)的数据读取请求发送到 CS1 上执行，而其余三个 Tablet 的数据读取请求分别发送到 CS2，CS3 和 CS4 上执行。

数据导出的性能取决于收到最多 SCAN 请求的 CS。因此，如何将 SCAN 请

求均匀发送到集群中的不同 CS 是提升数据导出性能的一个重要因素。一方面，这有利于充分利用集群中所有 CS 的资源。另一方面，减轻数据导出对其它应用程序查询请求的影响。CEDAR 自身由集群中 RS 负责负载均衡，它会定期选择某些 Tablet 从负载均衡较高的机器迁移到负载均衡较低的机器。因此，只要我们做好 SCAN 请求的负载均衡，集群最终的状态一定也是负载均衡的。

3.3.1 基于 Tablet 的负载均衡算法

在 CEDAR 中数据是以 Tablet 为单位分布在各个 CS 上，为了使数据导出过程中集群中的每个 CS 的负载大致相同，我们要将 Tablet 的 SCAN 请求均匀的分发给集群中的不同 CS 上。因为 Tablet 的大小是大致相等的，所以我们可以根据每个 CS 接收到 SCAN 请求数量作为负载均衡的判断标准。因此，我们提出了一种基于 Tablet 的负载均衡算法。

求解最优化问题的算法通常需要经过一系列的步骤，在每个步骤都面临着多种选择。为了使负载均衡算法更简单、更高效，我们借鉴了贪心算法的思想。贪心算法在每一步都做出当时看起来最佳的选择。也就是说，他总是做出局部最优的选择，寄希望这样的选择能导致全局最优解。

在 CEDAR 中每个 Tablet 默认有三个副本，则 ES 可以选择将 Tablet 的 SCAN 请求发送给其中任一个副本所在的 CS。我们将 CS 的负载均衡问题抽象成求最优解的问题。ES 每次发送 SCAN 请求时，都发送给负载最小的 CS，这就是说每次的选择都是最佳的。因此，当 ES 发送完所有的 SCAN 请求后，我们可以假定得到了最好的结果或者近似最好的结果。

该算法的关键在于记录集群中所有 CS 的负载情况。我们在 ES 端维护一个整型数组 cs_load[n]，用于记录每个 CS 接收到的 SCAN 请求数量。该数组的大小等于集群中 CS 的数量，每个下标对应一个 CS。每次 ES 在发送 SCAN 请求前，首先根据 Tablet 三个副本的分布信息读取 cs_load[n]对应下标的三个值，然后将 SCAN 请求发送给值最小的那个 CS。算法具体如下：

基于 Tablet 的负载均衡算法
输入：整型数据 cs_load[n]，Tablet 信息集合 输出：集群中 n 个 CS 负载情况 1：用 0 初始化数组 cs_load[n]，每个下标对应着 1 个 CS。 2：While Tablet 信息集合不为空{ 3： 取一个 Tablet 信息，读取三副本的分布信息为 CSx，CSy 和 CSz。然后比较 cs_load[x-1]，cs_load[y-1]和 cs_load[z-1]的值，若 cs_load[x-1]最小，则将该 Tablet 的 SCAN 请求发送给 CSx 并使 cs_load[x-1]的值加 1。} 4：返回 cs_load[n]

在 3.2.1 节的步骤二中有两点需要改动。第一点是，当 ES 接收到 RS 返回的 Response 信息后，需要根据 CS 的信息初始化一个整型数组 cs_load[n]，其中 n 等于 CS 的数量。第二点是，ES 构造好 SCAN 请求后，要根据 Tablet 的分布信息去访问 cs_load[n]并找出负载最低的 CS，最后将 SCAN 请求交给该 CS 对应的连接线程处理。下面以图 3-3 为例，讲述该算法具体的执行步骤：

CEDAR 集群中有四个 CS，则 ES 初始化一个大小为 4 的整型数组 cs_load[4]，其中(CS1, CS2, CS3, CS4)分别对应着数组下标(0,1,2,3)。首先，读取 Tablet(1)的信息获取其位置信(CS1, CS3, CS4)。其次，比较 cs_load[0]、cs_load[2]和 cs_load[3]的值，三个值均为 0，则默认选择第一个，也就是将 Tablet(1)的 SCAN 请求发送给 CS1。最后，将 cs_load[0]的值加 1。ES 再读取 Tablet(2)的信息获取其位置信息(CS2, CS3, CS4)。然后，比较 cs_load[1]、cs_load[2]和 cs_load[3]的值，ES 选择将 Tablet(2)的 SCAN 请求发送给 CS2。最后将 cs_load[1]的值加 1。同理，将 Tablet(3)、Tablet(4)和 Tablet(5)的 SCAN 请求发送给负载最小的 CS。因此，最终结果是 Tablet(1)和 Tablet(4)的 SCAN 请求被发送给 CS1，而 Tablet(2)、Tablet(3)和 Tablet(5)的 SCAN 请求分别被发送给 CS2，CS3 和 CS4。

3.3.2 基于微分片的负载均衡算法

为了使 CS 达到更好的负载均衡效果，我们提出了一种基于微分片的负载均

衡算法。其中，微分片是指比 Tablet 更小的范围。根据负载的需求，从 Tablet 中划分出相应比例的范围。微分片中除了起始主键和结束主键与 Tablet 不同之外，其余全部相同。下面我们介绍微分片的划分规则，目前只支持数值型单一主键的 Tablet。

首先，我们定义 tiny_start_key 和 tiny_end_key 分别为微分片的起始主键和结束主键是。接着，我们定义 tablet_start_key 和 tablet_end_key 分别为 Tablet 的起始主键和结束主键。我们用(tiny_end_key - tiny_start_key)与(tablet_end_key - tablet_start_key)的比值表示微分片范围占 Tablet 的比例。最后，用 last_key 记录微分片的结束主键，作为下一个微分片的起始主键。下面介绍基于微分片的负载均衡算法的主要思想：

1. 统计所有 Tablet 的数量 m 和集群中可用的 CS 数量 n。
2. 根据统计信息计算出每个 CS 的负载平均值 AVG=m/n。其中，AVG 表示 CS 接收到的多少个 Tablet 的 SCAN 请求。如果 AVG 不是整数，说明 CS 接收到的请求中存在占 Tablet 相应比例的微分片的 SCAN 请求。另外，只会为 AVG 保留一位小数。
3. 维护 n 个集合用于统计每个 CS 上有哪些 Tablet，其中每个 CS 对应一个集合。根据每个 Tablet 的分布信息，将它添加到对应的集合中。当所有 Tablet 都已经添加到对应的集合中时，对每个集合中的 Tablet 按照起始主键进行排序。
4. 为每个集合构建一个复合主键，它就是该集合中所有 Tablet 的起始主键的集合。并按照复合主键对 n 个集合进行排序。
5. 维护两个浮点型数组，一个是 left_percent [m]用于记录 Tablet 的待请求比例，另一个是 got_scan[n]用于记录每个 CS 已经接收到 Tablet 的 SCAN 请求数量。
6. n 个集合对应 n 个 CS，对排好序的数组顺序处理。同样，顺序处理每个集合中的 Tablet。对每个集合进行处理前，首先访问 got_scan[n]相应下标的值。如果该值等于平均负载 AVG，则表示该集合对应的 CS 负载已经分配完毕，接着处理下一个集合。

7. 对集合中的 Tablet 进行处理时，首先访问 left_percent[m]数组对应下标的值 x 和 got_scan [n]数组对应下标的值 y。如果 x<=y，则根据 x 的值和该 Tablet 中的 last_key 构造 SCAN 请求，然后发送给该集合对应的 CS，最后更新两个数组中相应下标的值。如果 x>y，则根据 y 的值和该 Tablet 中的 last_key 构造 SCAN 请求，然后发送给该集合对应的 CS，最后更新两个数组中相应下标的值。
8. 直到处理完所有的集合算法结束。

基于微分片的负载均衡算法

```
输入：left_percent[m]，got_scan[n]，n 个集合 set[n]， AVG
输出：SCAN 请求均匀发送给各个 CS
1：对 n 个集合中的 Tablet 按照其起始主键排序
2:为 n 个集合构建复合主键,并按复合主键对 n 个集合进行排序得到 CS_SET
3：while CS_SET 不为空
4：     取出 CS_SET 中最小的集合 set[p]，其中 0<=p<n
5：     while set[p]不为空且 got_scan[p]<AVG
6：         取 p 中最小的元素 Tablet[q]，其中 0<=q<m
7：         if left_percent[q]==0
8：             continue //已发过 Tablet[q]的全部 SCAN 请求
9：         else if left_percent[q]< AVG- got_scan[p] then
10：            got_scan[p]+= left_percent[q]
11：            left_percent[q]=0 //q 的 SCAN 请求发送给 CS[p]
12：        else //划分 Tablet[q]相应比例构造并发送 SCAN 请求
13：            left_percent[q]-= (AVG-got_scan[p])
14：            got_scan[p]=AVG //CS[p]负载已满，取下一个集合
15：算法结束，所有 Tablet 的 SCAN 请求均匀的发送给各个 CS
```

为了使上述算法更易于理解，下面以图 3-3 为例介绍该算法。图中展示了在 CEDAR 集群中有 5 个 Tablet，它们分布在 4 个不同的 CS 上。下面具体描述基于微分片的负载均衡算法的执行流程：

步骤一：首先，计算CS的平均负载AVG=Tablet数量/CS数量，即AVG=1.25。其次，用1初始化一个浮点型数组left_percent_tablet[5]。其中，数组的每个下标对应着一个Tablet，初始值均为1。第三，初始化一个浮点型数组recrived_scan_cs[4]，每个下标对应着1个CS，初始值均为0。第四，为每个Tablet维护一个last_key记录划分出的微分片的结束主键，初始值为Tablet起始主键。

步骤二：统计所有CS上的Tablet信息得到4个集合，接着对每个集合中的Tablet按照起始主键排序，排序结果如下：

CS1	SET1{1，3，4，5}
CS2	SET2{2，4，5}
CS3	SET3 {1，2，3，4}
CS4	SET4{1，2，3，5}

步骤三：为4个集合构建复合主键，复合主键由集合中所有Tablet的起始主键构成。接着对集合CS_SET{ SET1，SET2，SET3，SET4}进行规则排序，排序结果如下：

CS3	SET3{1，2，3，4}
CS4	SET4{1，2，3，5}
CS1	SET1{1，3，4，5}
CS2	SET2{2，4，5}

步骤四：负载分配：从排好序的CS_SET集合中选择SET3{1，2，3，4}集合。接着从SET3集合中选择Tablet(1)。此时left_percent[0]<AVG-got_scan[2]，则将Tablet(1)的SCAN请求发送给CS3。然后，更新Tablet的last_key值为结束主键，更新got_scan[2]=1，以及更新left_percent[0]=0，到此Tablet(1)已处理完成。

接着从SET3中选择Tablet(2)。此时left_percent_tablet[1]> (AVG-got_scan[2])，则从Tablet(2)中划分出占比0.25的微分片后构造SCAN请求发送给CS3。然后，更新Tablet的last_key值为微分片的结束主键，更新left_percent[1]=0.75，以及更新got_scan[2]=1.25，到此CS3的所有负载已经分配完成。

从 CS_SET 中选择 SET4。然后，从 SET4 集合中选择 Tablet(1)。此时发现 left_percent[0]=0，说明该 Tablet 的 SCAN 请求已经发送给了其它 CS。无需再对 Tablet(1)进行处理，然后从 SET4 中选择 Tablet(2)。此时 left_percent [1]=0.75，那么说明 Tablet(2)还有剩余的相应比例的 SCAN 请求未处理。此时，left_percent[1]< (AVG-got_scan[3])，则可以将 Tablet(2)剩余比例的微分片 SCAN 请求全部发送给 CS4。然后，更新 got_scan[3]=0.5，以及更新 left_percent[1]=0。到此 Tablet(1)已处理完成。

接着从 SET4 中选择 Tablet(3)。此时 left_percent[2]>(AVG-got_scan[3])，则从 Tablet(3)中划分出占比 0.5 的微分片后构造 SCAN 请求发送给 CS4。然后，更新 Tablet 的 last_key 值为微分片的结束主键，更新 got_scan[3]=0，以及更新 left_percent[2]=0.5。到此 CS4 的负载已经分配完成。重复上述步骤，顺序处理 SET1 和 SET2，处理完所有的集合后算法结束。所有 Tablet 的 SCAN 请求发送完毕，且每个 CS 接收到的请求数量大致相同。

3.3.3 分析与讨论

本节提出了两种负载均衡算法。第一种是基于 Tablet 的负载均衡算法，算法的基本思想同贪心算法。在每一步都做出当时看起来最佳的选择，则经过一系列的选择后，最终能得到最优的结果或者是近似最优的结果。该算法以 Tablet 为最小负载单位，每次将 SCAN 请求发送给三个副本所在的 CS 中当前负载最小的一台。算法简单高效，但是并不能做到完全的负载均衡，就是说不同的 CS 接收到不同数量 Tablet 的 SCAN 请求。

为了达到更好的负载均衡效果，使所有 CS 上的负载几乎相同。我们考虑是否可以将 Tablet 划分成更小的分片，于是就提出了基于微分片的负载均衡算法。首先，先根据 Tablet 的分布信息计算出每台 CS 的平均负载 AVG。然后，将 Tablet 划分为更小的微分片，以微分片为单位向 CS 发送 SCAN 请求。最终达到了更好的负载均衡效果。

3.4 本章小结

本章主要介绍了面向静态数据的数据导出技术及其优化算法。首先，指出传统数据导出的方法中存在的问题，通过分析 CEDAR 数据库的架构和底层存储的特点，找到可以优化的点。然后，我们设计并实现了一种面向静态数据的导出方法，通过 ES 与 CS 直接交互，减少了网络传输次数，该方法支持数据导出并发执行，也解决了内存溢出的问题。最后，对新的方法做了一些优化，使其能充分利用集群中的基线数据服务器资源，同时也做到了负载均衡。

第四章 动态数据捕获

在第四章我们提出了面向静态数据的数据导出方法，用于支持快速高效获取快照数据。但是，为了获取更新版本的数据只能等待集群完成每日合并更新静态数据的版本或者主动请求集群开始每日合并操作。本章首先分析 CEDAR 的动态数据的管理机制，其次分析获取动态数据的可行性，最后提出了两种获取动态数据的方法，最后详细介绍它们的具体实现。

4.1 概述

通常在数据量较少的情况下，我们从源数据库将全部数据复制到目标数据库时选择的策略是：先将目标数据库的数据清空，然后从源数据库导出全部数据在导入到目标数据库中。这种方法简单有效，但是会对源数据库的性能有一定的影响。此外，如果数据规模很大，每一次都采用这种策略进行数据复制，会耗费大量的物理资源和时间资源。自上次全量导出以来，在源数据库上可能只发生了少量的更新，或者相比于历史数据来说更新数据的规模很小很小。因此，我们考虑对于已经有过一次全量导出的源数据库，只获取其增量更新数据。

在分布式数据库 CEDAR 中静态数据与动态数据分离存储，分别存储在基线数据服务器 ChunkServer 和更新服务器 UpdateServer 上。其中 UPS 是集群中唯一接收写操作的模块，所有的更新操作都会被写入 UPS 的内存表 Memtable 中。当 Memtable 的大小超过某个阈值时，UPS 会冻结内存表并创建新的内存表，此时被冻结的内存表会生成快照文件并存储到 SSD[]中。为了保证可靠性，UPS 在每次执行更新操作时，都会产生一条 commit_log 日志追加到已有的日志序列之后，即每次更新 Memtable 之前都会进行写日志的操作。当服务器发生宕机并重新启动后会先加载快照文件，接着将快照点之后的日志进行回放，使其恢复到与宕机之前相同的状态。每次冻结的 Memtable 转储到 SSD 之后，会进行切换日志

文件操作，就可将日志回放点移动到冻结内存表之后的日志文件，不仅释放了内存空间，同时也大大减少了日志回放数量。

由上文可知，CEDAR 数据库的动态数据有两种存在形式：一是 Memtable，二是提交日志 commit_log。要想获取增量数据，只要获取其中之一即可。若是获取 Memtable 内容，则需要将 Memtable 与静态数据融合后才能得到最新结果。若是获取日志文件，则需要在静态数据的基础上进行日志回放操作。

上一章讲述面向静态数据的数据导出，静态数据分布在 CS 上，该数据是上一次每日合并的结果。此时 UPS 的内存表中可能存储着大量的增量更新数据，若是仅导出 CEDAR 集群中的静态数据，则需要在此基础上回放大量的操作日志才能达到与源数据库的数据一致。因此，为了降低应用更新的代价，首先要获取最新版本的数据。我们对上一章的数据导出方法做一些修改，使其获取的数据为静态数据与最新版本的内存表融合后的数据，并在此基础上提出了一种基于日志的动态数据捕获方法。

4.2 最新版本的数据导出

为了获取最新版本的数据，不仅需要 CS 上的静态数据，而且还要获取 UPS 内存表中的增量更新数据。在 CEDAR 的查询执行流程中，首先查询节点 MS 会把请求发送给所有请求涉及到的 Tablet 所在的 CS，接着 CS 读取本地的静态数据并向 UPS 请求相应的增量更新数据，最后 CS 将静态数据与增量更新数据融合后返回给 MS。

在 CEDAR 中，为了使 UPS 的内存使用量保持在一定的阈值内，会定期执行每日合并操作。首先，UPS 会在系统设定的时间向 RS 发起每日合并请求。其次，UPS 冻结当前内存表并创建新的内存表，此时所有的增量更新数据会写入新的内存表中。第三，RS 接收到 UPS 的请求发后，通知 CS 执行每日合并操作。第四，CS 向 UPS 请求相应的增量更新数据，接着与本地的静态数据融合生成新版本的静态数据。最后，CS 向 RS 汇报新版本的 Tablet 信息。CEDAR 的每日合并操作会耗费系统大量的资源，且整个过程持续较长时间。因此，在第三章的基

础上我们提出了最新版本的数据导出方法。

4.2.1 基本思想

图 4-1 展示了最新版本的数据导出方法整体执行流程。首先，导出服务器 ExportServer 向 RS 发起导出最新版本数据的请求，RS 接收到 ES 的请求后会返回当前 Memtable 的版本号 mem_version。其次，ES 向 CS 发起带有最新数据版本号的 SCAN 请求。第三，CS 接收到 SCAN 请求时，发现本地静态数据的版本号落后于请求中的版本号，则 CS 会向 UPS 请求相应版本的增量更新数据。最后，CS 将静态数据和增量更新数据融合后返回给 ES。该方法中有些步骤与 3.2 节的大致相同，下面详细介绍执行流程中的不同之处：

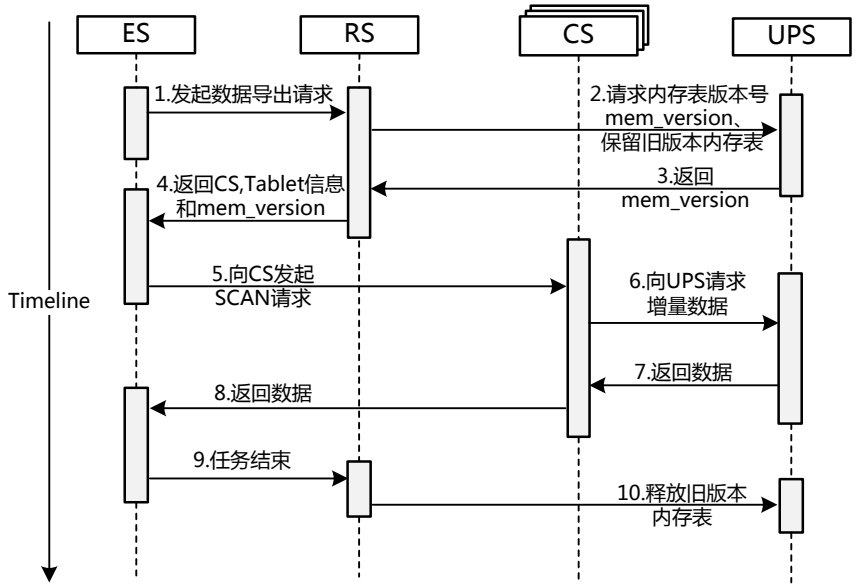

图 4-1 最新版本数据导出流程

步骤一：ES 向集群中的主控节点 RS 发起最新版本数据导出请求。

步骤二：RS 向 UPS 请求当前 Memtable 的版本号 mem_version，并通知 UPS 保留旧版本的 Memtable。

步骤三：UPS 查询 Memtable 的版本号 mem_version 并返回给 RS。

步骤四：RS 接收到 UPS 返回的信息后，构造 Response 信息，包括 mem_version、集群中可用 CS 信息和 Tablet 信息。

步骤五：ES 接收到 Response 信息后，进行本地的一些初始化工作。然后向

CS 发起带有 mem_version 参数的 SCAN 请求。

步骤六：CS 收到 ES 的 SCAN 请求后，发现本地静态数据版本小于 mem_version，则向 UPS 请求相应的增量更新数据。

步骤七：UPS 扫描内存表将对应 mem_version 版本的数据返回给 CS。

步骤八：CS 将静态数据和 UPS 返回的 mem_version 版本增量更新数据融合后返回给 ES。

步骤九：ES 接收到所有 CS 返回的数据时，通知 RS 任务结束。

步骤十：RS 通知 UPS 可释放旧版本的内存表。

4.2.2 分析与讨论

在 CEDAR 中存在每日合并操作，当 UPS 上的 Memtable 的大小超过一定值时会冻结 Memtable 并创建新的 Memtable，然后 UPS 会向 RS 请求执行每日合并操作。当每日合并操作执行完成后 UPS 会删除冻结的 Memtable，因此当 RS 接收到 ES 发来的数据导出请求时，会根据请求的参数类型决定是否通知 UPS 保留旧版本的 Memtable。

这个方法是在面向静态数据的导出方法基础上进行改进的，通过 CS 向 UPS 请求最新版本的增量更新数据。虽然，该方法多了一次网络传输时间，但是，大大降低了日志传输量和应用更新的代价。此外，为了持续的获取增量更新数据，我们在下一节介绍基于日志的动态数据捕获方法。

4.3 基于日志的动态数据捕获

从源数据库导出数据时，每次都采用全量导出的方式，使该数据源达到最新的状态，这将耗费大量的时间。若是将数据导出的方式变成增量导出，获取自上次导出的数据后发生变化的数据或者数据变化序列，也可以使数据源达到最新的状态。采取增量导出的前提是实现数据源的初始化，即一次全量数据的导出。增量导出的实现就是高效准确地获取自上次数据导出后的变化数据或者变化序列[46,47]。获取增量更新的方式包括三种：即触发器法、快照差分法和日志分析法。

触发器[48]是数据库系统在特定的条件或事件发生时调用的存储过程。常见

的事件包括插入、修改和删除等。触发器法是指在源数据库系统的表上建立相应的触发器。对表的数据进行更新操作会立即激活触发器将变化的数据写入一个临时表中。虽然，它捕获变化数据的性能很高。但是，一方面编写触发器需要很大的开发和维护成本，另一方面对数据库系统的性能会产生比较大的影响。

快照差分法[49]是通过比较两个时间点的数据快照来获取增量数据的。快照差分法有两个比较经典的实现算法：一是 Sort Merge 快照差分法，二是 Partition Hash 快照差分法。本质上来说两种算法都需要对两份快照数据进行逐条记录的比较。快照差分法对源数据库系统的实现没有依赖，通用型较强。但是，每次进行增量检测时都需要整个扫描新快照和旧快照，检测效率低下。随着数据量的增大，扫描整个快照会成为瓶颈。

日志分析法[50]就是通过分析数据库日志中的信息来捕获数据的变化序列。通过监控数据库的日志变化，并对这些日志进行有效的分析，则可以直接获取到所需的增量数据。这种方法不会占用太多额外的系统资源，对数据库的性能基本不会有影响，而且能在较短的延迟内就获取到相应的增量数据。但是，不同的数据库系统日志格式和接口在具体细节上存在很大差异，需要为每一种类型的数据库开发特定的日志解析程序。

显然，针对 CEDAR 数据库使用基于日志分析法来获取增量更新数据是最为合适的。CEDAR 是一款开源的分布式数据库系统，所以我们可以很方便的了解日志的具体格式和相应的接口。因此，我们提出了基于日志的动态数据捕获方法。通常，在一个繁忙的业务系统中，对数据库中表的同一元组可能会进行频繁的更新操作，导致日志量远大于其中实际包含的真实数据。为了降低应用更新的代价，减少对相同元组的冗余操作，我们在日志解析过程中对相同元组的一组操作进行了精简优化处理。

4.3.1 基本思想

在 CEDAR 中，UPS 是集群中唯一接受写入的模块，且 UPS 在更新内存表之前都需要写操作日志。因此，只要获取 UPS 上的操作日志，就相当于获取了所有数据的变化序列。此外，当内存表的数据量超过一定值时，CEDAR 会冻结

内存表生成快照文件并转储到 SSD 中。在 CEDAR 中，快照点之前的操作日志默认是可以删除的。

基于日志的动态数据捕获方法的主要思想是：首先，ES 向 RS 发起最新版本的数据导出请求。其次，RS 向 ES 返回当前 Memtable 对应的日志号 last_lsn。第三，ES 主动向 UPS 发起注册请求，只要注册成功，UPS 在写操作日志时会将日志同步给 ES。第四，ES 根据 UPS 发来的操作日志和之前 RS 返回的 last_lsn，主动向 UPS 请求缺失的日志。图 4-2 展示了基于日志的动态数据捕获方法的整体执行流程。该方法是与 4.2 节的方法相结合的，在请求导出最新版本数据的同时，并请求持续性获取此数据版本之后所有的更新操作日志。下面详细介绍获取更新操作日志的流程：

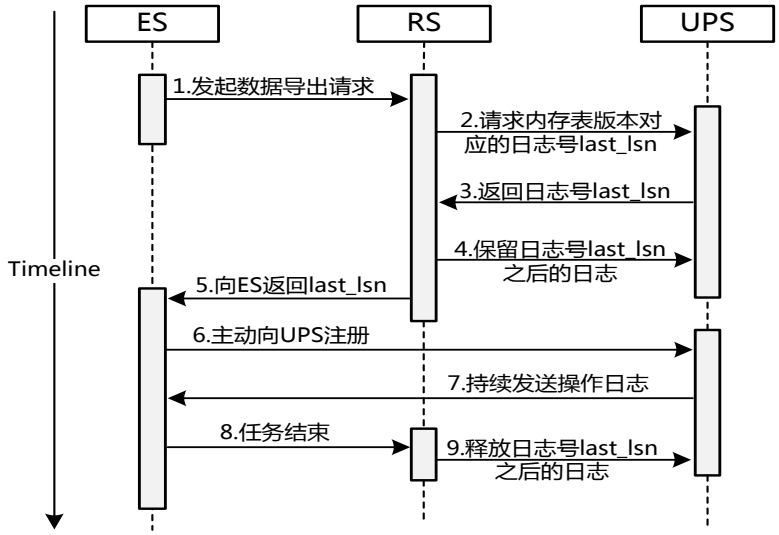

图 4-2 动态数据捕获执行流程

步骤一：ES 向集群中的主控节点 RS 发起导出最新版本数据的请求。

步骤二：RS 向 UPS 请求当前 Memtable 的版本号 mem_version 对应的日志号 last_lsn。同时，RS 通知 UPS 保留旧版本内存表对应的日志回放点之后的日志。

步骤三：UPS 向 RS 返回 mem_version 版本对应的日志号 last_lsn。

步骤四：RS 通知 UPS 保留日志号 last_lsn 之后的日志。

步骤五：RS 向 ES 返回 last_lsn。

步骤六：ES 主动向 UPS 注册，请求持续性获取操作日志。

步骤七：UPS 写操作日志并向 ES 发送该日志。

步骤八：ES 接收到所有 CS 返回的数据时，通知 RS 任务结束。

步骤九：RS 通知 UPS 可释放日志号 last_lsn 之后的日志。

4.3.2 实现细节

在保证正确性的前提下，实现持续的从 UPS 获取写操作日志的关键技术在于：日志同步、日志拉取和日志精简。在一个繁忙的业务系统内，可能对数据库中表的同一元组进行频繁的更新操作。在回放日志时，对同一元组进行多次更新操作，这就出现了冗余的操作。因此，我们在日志解析的过程中对这类操作进行优化，将同一元组的多次更新操作压缩成一次更新操作，称为日志精简。

日志同步：首先，ES 主动向 UPS 发起注册请求。注册成功后，UPS 上有一个专门的线程负责读取操作日志加入到缓存区中并发送给 ES。这里采用成组传输的方式，将多个写操作日志组成一批，一次性发给 ES。有两个条件决定了 UPS 何时将日志同步给 ES：一是时间阈值 T，二是日志数量 N。假设尝试从缓存区中读取 N 条操作日志，如果缓存区中的日志数量未达到 N，则等待一段时间 T，直到读取到 N 条日志或者超时为止。

日志拉取：ES 端设有一专门的日志缓存区用于接收 UPS 同步的日志。ES 的接收线程接收日志，并写入日志缓存区。如果发生异常情况，例如网络异常或者 UPS 重新启动，日志缓存区中没有日志或者日志不连续，此时 ES 会主动向 UPS 请求拉取操作日志。缺失的日志可能是网络不稳定造成的，经过较大的延迟时间后才发送到 ES，若是 ES 接收到重复的日志，会直接丢弃不做其他处理。值得注意的是，当 ES 第一次接收到 UPS 发送来的日志时，将该日志的日志号 first_lsn 与从 RS 返回的 last_lsn 进行比对，若是 first_lsn>last_lsn，那么 ES 会主动向 UPS 请求拉取缺失的日志。

日志精简（Log Compaction）：ES 的日志解析线程会从日志缓存区中读取操作日志，在内存中解析日志并写入本地的回放操作文件中。在一个事务内可能会对同一元组进行多次修改操作，其中多次操作是冗余的。因此，通过日志解析，

同一元组对象的多次更新操作合并为一次，这减少了最终生成的文件大小，并且还减少了应用更新的代价。在 CEDAR 中，每一个元组都存在主键，因此可以使用主键来区分不同的元组，对事务内的 Insert，Update 和 Delete 操作做精简优化处理。为了表示在一个事务内对一个元组的操作过程，用 I(?)，U(?)，D 三个符号分别表示 Insert，Update，Delete 操作。图 4-3 展示了一个事务内对同一元组的连续两次更新操作经过精简优化的结果。

	第二次操作			
第一次操作	操作	I(value2)	U(value2)	D
	I(value1)	不存在	I(value2)	NULL
	U(value1)	不存在	U(value2)	D
	D	不存在	不存在	不存在

图 4-3 更新操作精简

通常，在一组操作中 I 和 D 只会出现一次，而 U 会出现多次。如图 4-3 所示，两次相同的操作，但执行的先后顺序不同，精简优化后的结果是不同的。例如，先执行插入操作，再执行更新操作，则最终的操作是插入操作。而先执行更新操作，再执行插入操作，通常这是不存在的，如果遇到这种情况，不做任何处理。此外，删除操作一般都是出现在一组操作中的最后。日志精简算法具体如下：

日志精简算法

输入：同一个元组的增删改操作集 op_set

输出：对元组的一个操作 result_op

```
1:result_op = NULL
2:while op_set 不空，取出一个操作 op{
3:   if result_op == NULL then        //给 result_op 赋予
4:     result_op = op                //初值
5:   else if result_op == I(value1){//插入操作后通常只出现
6:     if op == U(value2) then      //更新操作，若出现其他操
```

```
7:       result_op = I(value2)        //作，直接退出
8:     else
9:       return }
10:  else if result_op == U(value1){//更新操作后可出现更和
11:    if op == U(value2) then       //删除操作，遇到插入操作
12:      result_op = U(value2)       //直接退出
13:    else if op == D then
14:      result_op = D
15:    else if op == I(value2) then
16:      return }
17:  else if result_op == D          //删除操作通常在一组操作
18:    return}                       //的最后，遇到其他操作直
19:return result_op                   //接退出
```

4.3.3 分析与讨论

采用日志分析法可持续获取增量数据且对数据库系统的性能很小，同时为了降低日志传输量和应用更新代价，我们在采用最新版本的数据导出方法来实现首次的数据全量抽取的前提下，开始基于日志的动态数据捕获方法。我们在 ES 内部实现了日志拉取和日志同步的功能，保证了数据的正确性。此外，在日志解析过程中采用日志精简算法进行优化，减少了对相同元组冗余操作，降低应用更新代价。

4.4 本章小结

本章主要介绍两种不同的数据导出方法及其实现。首先，介绍 CEDAR 数据库中增量数据的两种存在形式，分别是内存表 Memtable 和操作日志 commit_log，并分析获取增量数据的可行性。然后，提出了最新版本的数据导出方法，最新版本数据指静态数据和内存表融合后的结果。通过与第三章的方法进行对比，分析各自的优缺点和适用场景。最后，提出了基于日志的动态数据捕获方法及其具体

实现。通过日志同步和日志拉取功能保证数据的正确性，并提出日志精简算法，减少对相同元组的冗余操作，降低应用更新代价。

第五章 实验

本章设计了一系列的实验，用于对比传统数据导出方法和本文提出的面向静态数据的导出方法的时间差异和对集群性能影响。此外，我们使 CEDAR 数据库运行在单行更新的负载下，对比加载和不加载导出服务器 ExportServer 时集群中更新服务器的资源使用情况。最后，对实验结果进行分析与总结。

5.1 实验配置

5.1.1 实验环境

本章描述的所有实验都是在 CEDAR（0.2 版本）数据库上进行的。本实验使用了 8 台具有相同软硬件环境的服务器，服务器的具体配置如图 5-1 所示。

处理器	Inter(R) Xeon(R) E5-2620 2.1GHZ 24核
内存	128GB
磁盘	3T SSD
以太网	千兆网卡
操作系统	CentOS release 6.5(Final) 64位

图 5-1 服务器配置说明

本章选取了 7 台服务器来搭建 CEDAR 单集群。其中，一台服务器用于部署 CEDAR 的主控服务器 RootServer 和更新服务器 UpdateServer，一台服务器用于部署合并服务器 MergeServer，剩余五台服务器用于部署基线数据服务器 ChunkServer。最后，还有一台服务器用于部署导出服务器 ExportServer。

5.1.2 实验对比方法

SQL-HINT：传统的基于 SQL 的数据导出方法。通过接口向 CEDAR 集群中

的 MS 发送带有 HINT 命令的 SQL SELECT 语句，查询静态数据。等到 MS 返回数据后，对数据进行格式化处理并写入文件中。

Tablet-ES：基于 Tablet 的数据导出方法。首先，ES 向 RS 请求 Tablet 分布信息，然后向 CEDAR 集群中的 CS 发起 Tablet 的 SCAN 请求，获取静态数据。ES 接收到 CS 返回的数据后，对数据记录格式化处理后写入本地文件。

Tablet-Balancing：基于 Tablet 的负载均衡算法。ES 向 CEDAR 集群中的 CS 发起 Tablet 的 SCAN 请求时，根据三副本的位置信息，将 SCAN 请求发送给负载最小的 CS。

SQL-NO-HINT：传统的基于 SQL 的数据导出方法。通过接口向 MS 发送 SQL SELECT 语句，查询最新版本数据。接收到 MS 返回的数据后，对数据进行格式化处理并写入文件中。

Memtable-ES：最新版本的数据导出方法。ES 根据 Tablet 的分布信息向 CS 发起带有内存表版本号 mem_version 的 SCAN 请求，请求最新版本的数据。当 ES 接收到 CS 返回的数据后，对数据记录格式化处理后写入本地文件。

Log-ES：ES 主动向 UPS 注册，注册成功后，每当 UPS 写操作日志时会将日志同步给 ES。若 ES 发现不连续日志或久未接收到日志，会主动请求相应日志。

5.1.3 基准测试

本实验使用的基准测试工具 YCSB 生成所需的测试数据集。其中，YCSB（Yahoo Cloud Serving Benchmark）全称是雅虎云服务基准，用于评估计算机程序的检索和维护功能的开源规范和程序套件。通常用于比较 NoSQL 数据库管理系统的相对性能，原来的基准是由 Yahoo！研究部门的工作人员在 2010 年发布的，其目标是“促进新一代云数据服务系统的性能比较”。它是为传统的数据库系统设计的基准，针对不同的数据库加载不同的交易处理工作负载。

YCSB 有两个重要的组件：YCSB 客户端和核心工作负载。YCSB 分为两种模式，分别是 load 和 run。其中，YCSB 客户端是一个可扩展的工作负载生成器，工作负载定义了在 load 阶段将要加载到数据库上的数据，我们利用它来生成测试数据，工作负载可以通过参数文件进行定义，下面介绍参数文件的几个重要参

参数	描述
Fieldcount	一条记录中的字段个数
Fieldlength	每个字段的大小
Insertproportion	插入比例
Table	表名
Recordcount	记录条数

图 5-2 基准测试参数表

数，具体配置如图 5-2。

本文利用 YCSB 生成测试数据集，其数据量为五千万条记录。数据表的 schema 如图 5-3 所示，共有 16 个字段，1 个字段为主键，主键类型为 int 型，其余 15 个字段为非主键，其中 5 个字段是 int 型，10 个字段为字符类型，大小是 100 字符，则每条记录大约是 1KB，即测试数据集大小约为 50GB。首先，我们向 CEDAR 数据库中导入四千万条记录，然后向 CEDAR 数据库发起每日合并请求，等到每日合并完成后，再向集群中导入一千万条记录，则此时集群中，CS 上的基线数据大小约为 40GB，UPS 上的增量数据大小约为 10GB。上一节提到的实验对比方法均在此环境下进行测试。

字段类型	字段数量	字段大小（Byte）
Int	6个	4
Varchar	10个	100

图 5-3 数据表的 schema

5.2 实验结果与分析

5.2.1 面向静态数据导出的实验结果与分析

图 5-4 展示了在集群中有三台 CS 的配置下，使用 SQL-HINT 和 Tablet-ES 两种方法导出静态数据的时间比较。SQL-HINT 是传统的基于 SQL 的数据导出方

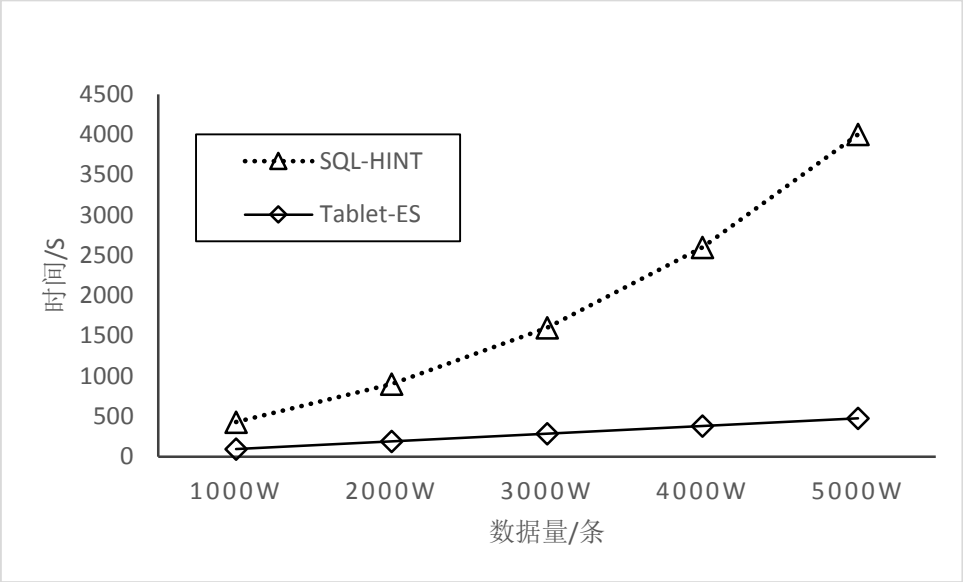

图 5-4 两种方法在数据量变化时的时间对比

法，Tablet-ES 是基于 Tablet 的数据导出方法。图 5-4 中，横坐标表示数据量，其单位是行，纵坐标表示数据导出时间，其单位是秒。从图 5-4 中可以看出 SQL-HINT 方法的时间始终大于 Tablet-ES 方法的时间，而且随着数据量的增长，它们之间的时间差越大。

SQL-HINT 方法是通过接口向 MS 发送 SQL 语句，然后 MS 解析 SQL 语句并将请求分发给相应的 CS。当 CS 返回所有 Tablet 数据时，MS 需要将所有返回的 Tablet 数据合并后才返回最终结果。而 Tablet-ES 方法是 ES 根据 Tablet 的分布信息，直接向 CS 发起相应 Tablet 的 SCAN 请求，ES 接收到 Tablet 的数据后就可以对其进行处理。相比于 SQL-HINT 方法，不仅减少了数据的一次网络传输时间，而且也不需要执行合并所有 Tablet 结果集的操作，只要有 CS 向 ES 返回数据，ES 就将数据放入缓存区中，有专门的 IO 线程将数据写入磁盘。此外，随着数据量的增长，两种方法耗费的时间差异越来越大。因为数据量越大，SQL-HINT 方法中 MS 需要等待所有数据从 CS 返回的时间越长，而且合并所有结果集的操作也越耗时，甚至当数据量达到一定程度时，MS 需要磁盘 IO 的操作。但是，Tablet-ES 方法是以 Tablet 为单位向 CS 发起请求的，无需等到所有数据返回再做处理。因此，Tablet-ES 方法的时间消耗与数据量接近于线性关系。

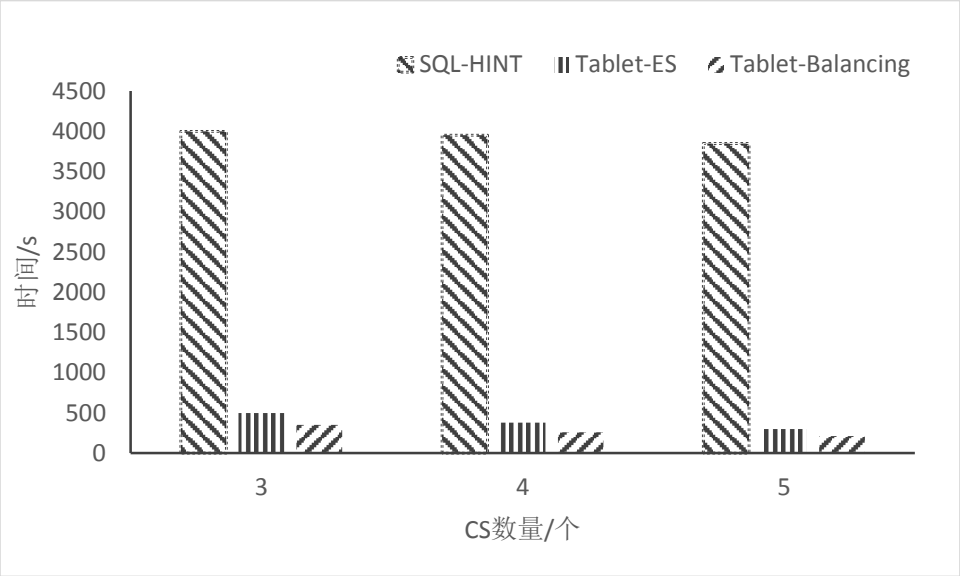

图 5-5 三种方法在 CS 数量变化时的时间对比

这组实验说明了 Tablet-ES 方法性能优于 SQL-HINT 方法，而且数据量越大它们之间的性能差异越大。

5.2.2 基于 Tablet 的数据导出的实验与分析

图 5-5 展示了在导出五千万条数据的条件下，三种方法的时间比较，一是 SQL-HINT 方法，二是 Tablet-ES 方法，三是 Tablet-Balancing 方法。SQL-HINT 方法是传统的基于 SQL 的数据导出方法，Tablet-ES 方法是基于 Tablet 的数据导出方法，Tablet-Balancing 方法是在 Tablet-ES 方法的基础上采用了基于 Tablet 的负载均衡算法的方法。图 5-5 中，横坐标表示 CS 的数量，其单位是个，纵坐标表示数据导出时间，其单位是秒。从图 5-5 中可以看出，SQL-HINT 方法的花费时间始终大于 Tablet-ES 方法花费的时间，Tablet-Balancing 方法花费的时间始终小于 ablet-ES 方法花费的时间。三种方法花费的时间都随着 CS 数量的增多而减少，SQL-HINT 方法时间下降的很少，而 Tablet-ES 方法和 Tablet-Balancing 方法的时间有明显的下降。

随着 CS 的数量增多，部分 Tablet 会被 RS 迁移到新的 CS 上。在 SQL-HINT 方法中，主要有两部分时间消耗，一是数据从 CS 传输到 MS 的时间，二是 MS 合并所有 CS 返回数据的时间。随着 CS 的增加，数据从 CS 传输到 MS 的时间

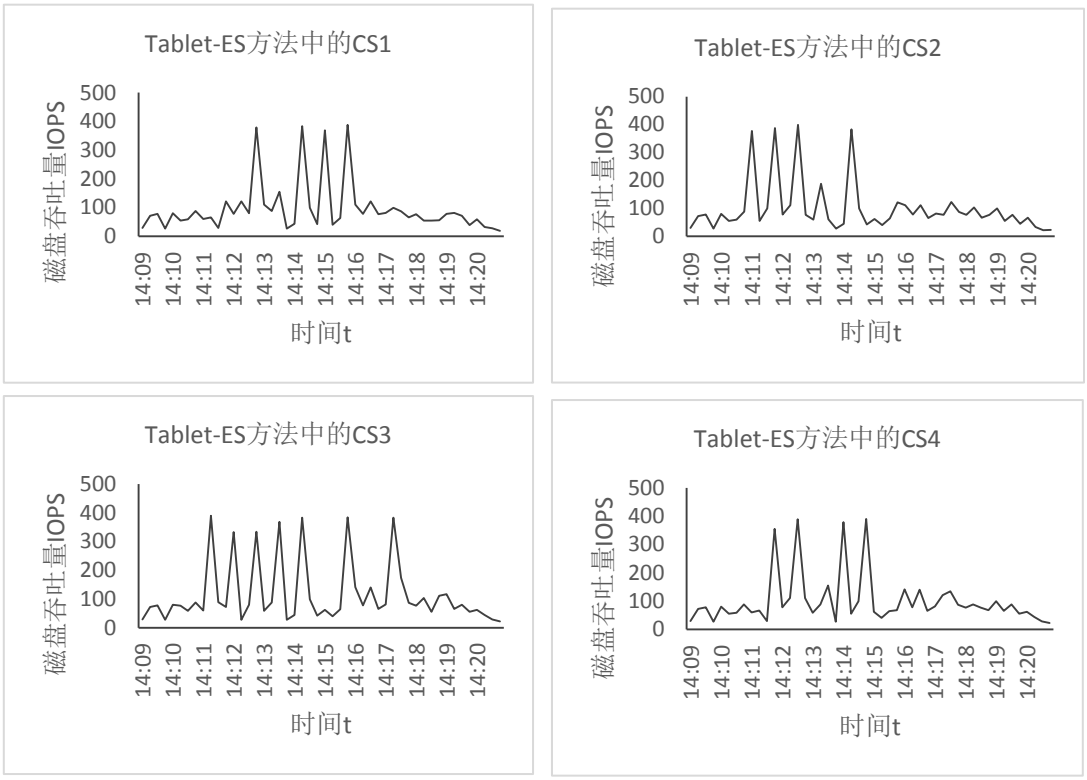

图 5-6 运行 Tablet-ES 方法时各 CS 吞吐量情况

减小了，但 MS 合并数据的时间没有减少，因此 SQL-HINT 方法时间略有下降，但并不明显。而 Tablet-ES 方法和 Tablet-Balancing 方法中，CS 的数量增多，那么 ES 端发送 SCAN 请求、接收数据和处理数据的并发度也相应的增加，因此时间消耗下降明显。

但是，在 Tablet-ES 方法中，每个 CS 接收到的 SCAN 请求数量不同，导致某些 CS 负载相对较高，而某些 CS 负载相对较低，所有任务完成时间取决于最慢的那个 CS。在 Tablet-Balancing 方法中，每个 CS 收到的 SCAN 请求数量大致相等，不会因单点的性能而影响整体，因此它的时间消耗小于 Tablet-ES 方法。

这组实验说明了 Tablet-ES 方法随着集群中 CS 数量的增长，其时间下降的趋势大于 SQL-HINT 方法。此外，负载均衡算法也有利于性能的提升。

5.2.3 基于 Tablet 负载均衡算法的实验与分析

图 5-6 展示了在集群中有四台 CS 的配置，使用 Tablet-ES 的方法导出五千万条数据时，集群中各 CS 的磁盘吞吐量情况。图 5-7 展示了在集群中有四台 CS

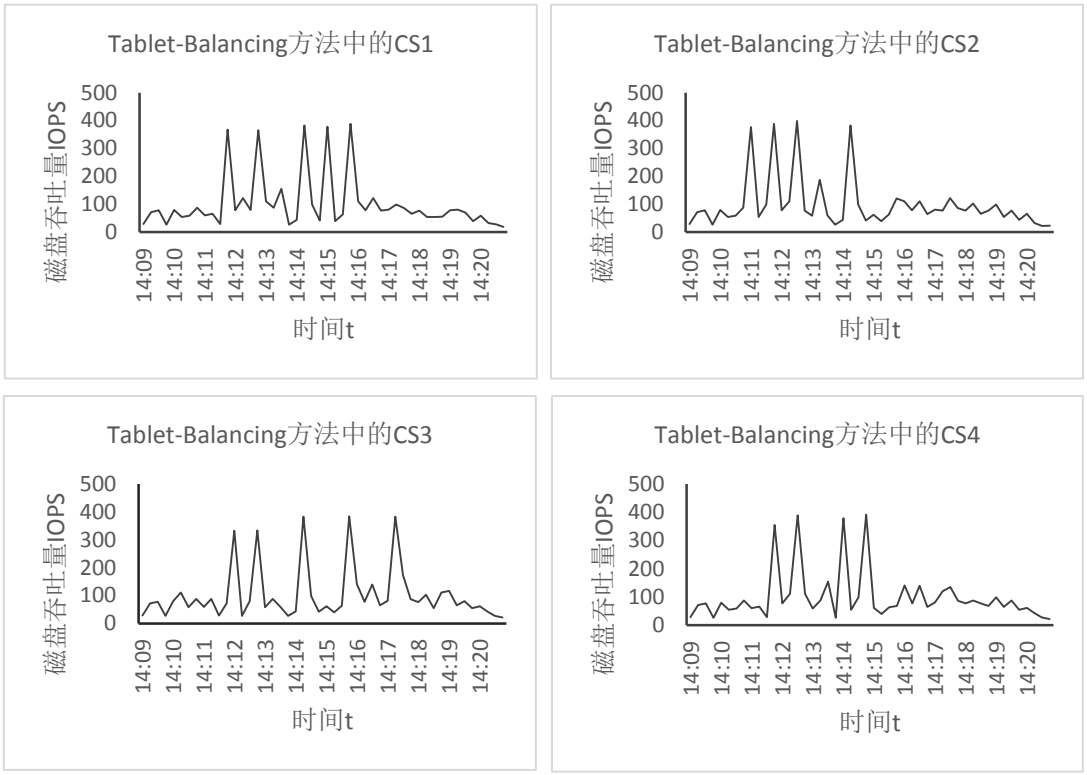

图 5-7 运行 Tablet-Balancing 方法时各 CS 吞吐量情况

的配置，使用 Tablet-Balancing 的方法导出五千万条数据时，集群中各 CS 的磁盘吞吐量情况。图中的横坐标都表示时间点，纵坐标都表示磁盘吞吐量，单位是 IOPS。图 5-6 展示了 Tablet-ES 方法运行期间各 CS 的磁盘吞吐量。图 5-7 展示了 Tablet-Balancing 方法运行期间各 CS 的磁盘吞吐量。本组实验通过 nmon 工具在两种方法执行期间，监控 CS 所在节点的磁盘使用情况，并根据监控的数据生成图 5-6 和图 5-7。

从图 5-6 中可以看出， CS3 的磁盘高负载工作持续时间最长， CS1，CS2 和 CS4 三者无太大差距。从图 5-7 中可以看出，CS1 和 CS3 磁盘高负载工作时间大致相同，略高于 CS2 和 CS4。它们的磁盘吞吐量开始变高的时间点不同，因为接收到 ES 发送的 SCAN 请求时间不同。当 CS 接收到 SCAN 请求时，会从磁盘中读取相应数据并返回给 ES。因此，磁盘的吞吐量可以说明每个 CS 的负载情况。CS 接收到 SCAN 请求数量越多，磁盘吞吐量的峰值会持续较长时间。由图 5-6 可知，CS3 接收到的 SCAN 请求最多，CS1、CS2 和 CS4 接收到的 SCAN 请求数大致相同。Tablet-ES 方法的导出数据的时间由 CS3 决定。由图 5-7 可知，

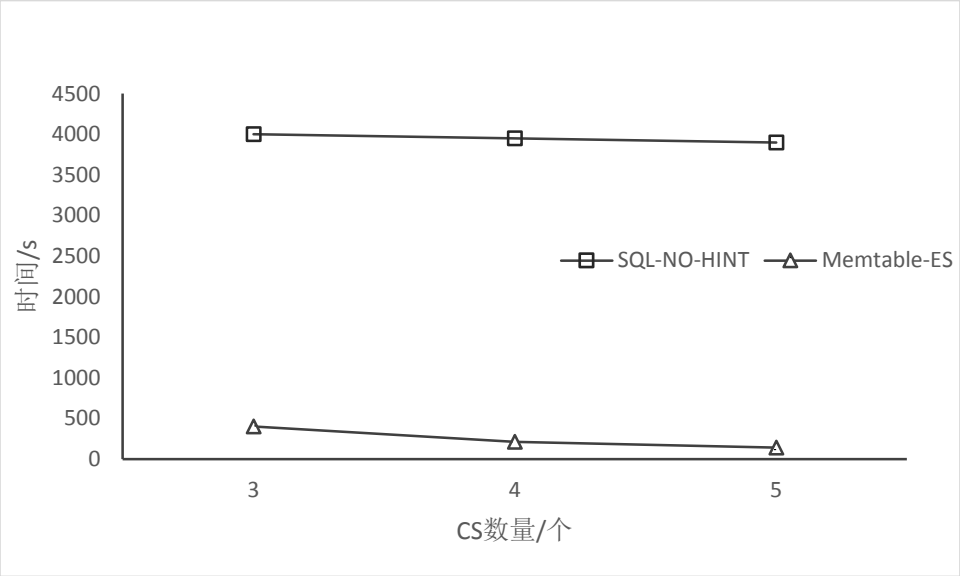

图 5-8 两种方法在 CS 数量变化时的时间对比

CS1 的负载相对于图 5-6 中变高了，而 CS3 的负载相对于图 5-6 中变低了。因为，原先发送给 CS3 的请求被发送给了 CS1。

这组实验说明 Tablet-Balancing 方法是有效的，和图 5-5 的实验共同说明了，负载均衡算法可以降低 CS 负载并提升性能。

5.2.4 最新版本的数据导出的实验与分析

图 5-8 展示了在静态数据量为四千万条，增量数据一千万条的条件下，两种方法导出全量数据的时间对比。一是 SQL-NO-HINT 方法，另一个是 Memtable-ES 方法。图中，横坐标是 CS 数量，单位是个，纵坐标是时间，单位是秒。从图中可以看出 Memtable-ES 方法花费的时间始终小于 SQL-NO-HINT 方法的。而且，随着 CS 数量的增长，Memtable-ES 方法花费时间的下降的要比 SQL-NO-HINT 方法快。因为 Memtable-ES 方法相比于 SQL-NO-HINT 方法，少了一次网络传输，且无需做合并所有 Tablet 数据的操作。此外，随着集群中 CS 数量的增加，在 ES 发送 SCAN 请求的并行度也在增加，那么 ES 接收和处理的速度也会提升。所以方法 7 所消耗的时间始终比方法 6 少，并且整体数据导出消耗时间下降的更为显著。

方法 1 是传统的基于 SQL 的数据导出方法，方法 2 是 ES 向 CS 发起指定内

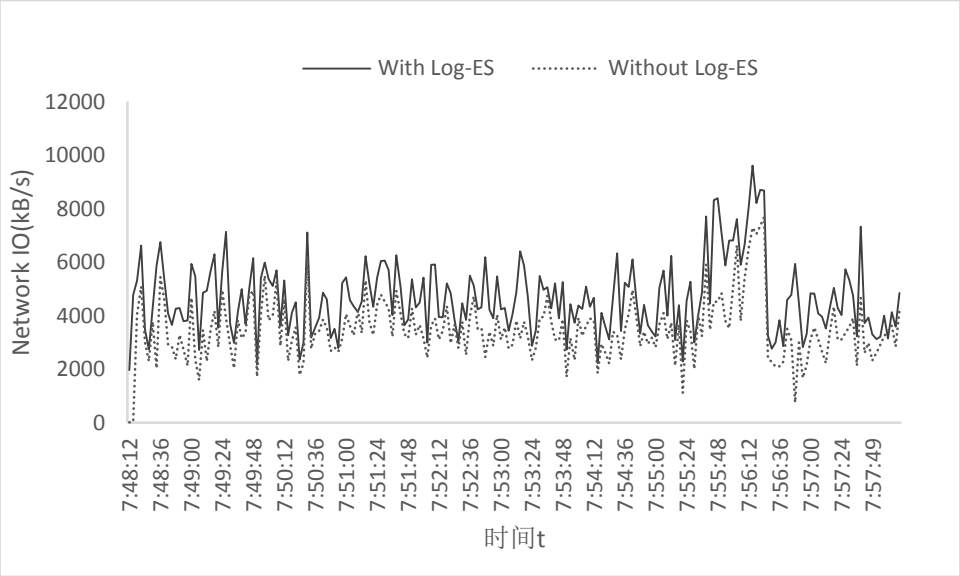

图 5-9 加载与不加载 Log-ES 时 UPS 的网络带宽

存表版本的数据导出方法。图 5-5 中，横坐标是 CS 的数量，单位是个，纵坐标是时间，单位秒。从图中可以看出，方法 2 所花费的时间始终小于方法 1 所花费的时间。此外，随着 CS 数量的增长，方法 2 花费时间的下降的要比方法 1 快。因为方法 2 比方法 1,少了一次网络传输,且无需做合并所有 Tablet 数据的操作。此外，随着集群中 CS 数量的增加，在 ES 发送 SCAN 请求的并行度也在增加，那么 ES 接收和处理的速度也会提升。所以方法 7 所消耗的时间始终比方法 6 少，并且整体数据导出消耗时间下降的更为显著。

5.2.5 基于日志的动态数据捕获的实验与分析

图 5-9 展示了 CEDAR 在单行更新负载下，运行与不运行 Log-ES 方法时，通过 nmon 工具监控集群中 UPS 的网络收发包的情况。图中，横坐标表示时间点，纵坐标网络的吞吐量，单位是 KB/s。从图中可以看出两条曲线非常的接近，这说明同步日志的操作占用 UPS 节点很少的网络资源，说明 Log-ES 对 UPS 基本没有影响。因为， UPS 上有一专门的线程负责读取日志并将日志加入到缓冲区中，批量将日志发送给 ES。此外，在 ES 端实现日志拉取功能，ES 主动请求缺失日志，大大简化了 UPS 端的工作。这组实验说明了基于日志的动态数据捕获方法对集群中的 UPS 基本没有影响。

5.3 本章小结

本章设置了大量的实验，对实验结果进行分析总结，从不同的方面去证明本文提出的几种数据导出方法的高效性。首先通过实验证明，基于 Tablet 的数据导出方法速度远远超过传统的基于 SQL 的数据导出方法，并对该方法提出了负载均衡算法，不仅充分了利用集群中各服务器的资源，同时也使数据导出的速度有了进一步的提升。另外，本章通过实验证明基于日志的动态数据捕获方法对源数据库的影响很小。

第六章 总结与展望

随着互联网的快速发展，各行业的业务数据都得到了爆发式的增长，如电信、金融等行业。数据是企业宝贵的资源，各企业都面临着海量数据存储与管理的问题。传统集中式数据库存在着明显的局限性，无法满足可扩展性、高性能、高可用的需求，为了应对这些挑战，一些企业开始尝试采用分布式数据库。分布式数据库以其良好的可扩展性受到工业界和学术界的关注，可以很好的替代传统的集中是数据库来解决海量数据的存储和管理问题。企业内部经常需要在不同系统之间进行大量的数据复制，这就需要从源数据库中抽取数据，因此如何从分布式数据库中将海量数据高效可靠的抽取出来是当前迫切需求。本文以分布式数据库 CEDAR 为研究对象，分析研究其整体架构和底层存储特点，提出了一种高效可靠的数据并行抽取方法。本文主要的工作如下：

1. 提出了考虑负载均衡的静态数据导出方法。通过导出服务器直接向基线数据服务器发起数据拉取请求，减少数据从基线数据服务到合并服务器网络传输时间，并且以 Tablet 分片为单位实现并发处理数据拉取请求，避免了合并 Tablet 结果集的操作，解决了占用大量内存的问题。
2. 提出了基于日志的动态数据捕获方法。在导出服务器内部实现日志拉取和同步功能保证了数据的正确性，而且在日志解析过程中精简对同一元组的多次操作，减少冗余操作，降低应用更新代价。
3. 通过实验，验证了我们提出的数据导出方法的可行性与高效性。我们在开源数据库 CEDAR 上实现了本文提出的数据导出方法。实验结果展示了本文提出的数据导出方法能有效的降低响应时间，减少系统资源占用。

综上所述，本文以分布式数据库 CEDAR 为研究对象，通过分析其架构和存储特点，解决了海量数据抽取的问题。同时，提出了几种不同的数据导出方法以应对不同的应用需求，并提出了相应的优化算法，在充分利用 CEDAR 集群各基线数据服务器资源的前提下，做到负载均衡。在基于日志的动态数据捕获方法中，

由日志同步和日志拉取保证数据正确性的前提下，实现持续性获取增量数据。

但是目前的实现还是存在一些不足之处，一是架构局限性：目前仅考虑读写分离的架构，其他分布式系统架构有待研究。此外，对数据的并发处理也是建立在数据库系统中数据横向切分的基础上。而且基于日志解析的增量捕获方法也需要读取日志的接口，并且并不是所有数据库都会对外公开日志的格式。二是实验仅采用标准的基准测试来模拟负载，可以在真实生产环境中进行测试。

参考文献

[1] Michael J Kamfonas. Recursive Hierarchies: The Relational Taboo. The Relational Journal, 27, 1992.

[2] Bachman C W. The programmer as navigator[J]. Communications of the Acm, 1973, 16(11):653-658.

[3] Kanellakis P C. Elements of Relational Database Theory[J]. Handbook of Theoretical Computer Science, 1991.

[4] Codd E F. A relational model of data for large shared data banks[M]// Pioneers and Their Contributions to Software Engineering. Springer Berlin Heidelberg, 2001:377-387.

[5] Koch G B, Loney K. Oracle 8: The Complete Reference[M]. Osborne/McGraw-Hill, 1997.

[6] Chamberlin D. Using the new DB2: IBM's object-relational database system[M]. Morgan Kaufmann Publishers Inc. 1996.

[7] Nielsen P, Parui U. Microsoft SQL Server 2008 Bible[J]. John Wiley & Sons, 2011.

[8] 李国杰, 程学旗. 大数据研究:未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考[J]. 中国科学院院刊, 2012, 27(6):5-15.

[9] Zsu M T, Valduriez P. Principles of distributed database systems[J]. Springer Berlin, 1999, 24(1):31-41.

[10] Corbett, James C, Dean, Jeffrey, Epstein, Michael, et al. Spanner: Google's globally-distributed database[C]// Usenix Conference on Operating Systems Design and Implementation. 2012:251-264.

[11] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics[J]. BMC Bioinformatics, 2010, 11 Suppl 12(12):S1.

[12]阳振坤.OceanBase 关系数据库架构.华东师范大学学报:自然科学版,(5):141-148,2014

[13]Kher V, Kim Y. Securing distributed storage:challenges, techniques, and systems[C]// ACM Workshop on Storage Security and Survivability, Storagess 2005, Fairfax, Va, Usa, November. DBLP, 2005:9-25.

[14]Wolfson O, Jajodia S, Huang Y. An adaptive data replication algorithm[J]. Acm Transactions on Database Systems, 1999, 22(2):255-314.

[15]王琳, 杨波, 高艳丽. Web2.0 互联网应用技术研究[J]. 中兴通讯技术, 2008, 14(5):14-17.

[16]Metzger B, Mauldin S, Sandell B, et al. Data migration: US, US 20070079140 A1[P]. 2007.

[17]Wolfson O. An adaptive data replication algorithm[J]. Acm Transactions on Database Systems, 1999, 22(2):255-314.

[18]Ranganathan K, Foster I. Identifying Dynamic Replication Strategies for a High-Performance Data Grid[J]. Lecture Notes in Computer Science, 2001, 2242:75--86.

[19]Hara T, Madria S K. Data Replication for Improving Data Accessibility in Ad Hoc Networks[J]. Mobile Computing IEEE Transactions on, 2006, 5(11):1515-1532.

[20]Conn S S. OLTP and OLAP data integration: a review of feasible implementation methods and architectures for real time data analysis[C]// SoutheastCon, 2005. Proceedings. IEEE. IEEE Xplore, 2005:515-520.

[21]Lima A A B, Furtado C, Valduriez P, et al. Parallel OLAP query processing in database clusters with data replication[J]. Distributed and Parallel Databases, 2009, 25(1):97-123.

[22]Stacey D. Replication: DB2, Oracle, or Sybase?[J]. Acm Sigmod Record, 1995, 24(4):95-101.

[23]Seidman C, Smith P. MySQL: The Complete Reference[M]. McGraw-Hill, Inc. 2003.

[24]杨传辉. 大规模分布式存储系统:原理解析与架构实战[M]. 机械工业出版社,

2013.

[25] On S T, Hu H, Li Y, et al. Lazy-Update B+-Tree for Flash Devices[C]// Tenth International Conference on Mobile Data Management: Systems, Services and MIDDLEWARE. IEEE, 2009:323-328.

[26] DataX. http://code.taobao.org/p/datax/wiki/. [Online;accessed 23-march-2017].

[27] 周婧, 王意洁, 阮炜,等. 面向海量数据的数据一致性研究[J]. 计算机科学, 2006, 33(4):137-140.

[28] Lee D, Mao W, Chiu H, et al. Designing Triggers with Trigger-By-Example[J]. Knowledge and Information Systems, 2005, 7(1):110-134.

[29] Lieuwen D F, Gehani N, Arlein R. The Ode active database: trigger semantics and implementation[C]// Twelfth International Conference on Data Engineering. IEEE Xplore, 1996:412-420.

[30] Shi J G, Bao Y B, Leng F L, et al. Study on Log-Based Change Data Capture and Handling Mechanism in Real-Time Data Warehouse[C]// International Conference on Computer Science and Software Engineering. IEEE Xplore, 2008:478-481.

[31] Ankorion I. Change Data Capture Efficient ETL for Real-Time BI.[J]. Dm Review, 2005(1).

[32] Mohan C, Haderle D, Lindsay B, et al. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging[C]// Readings in Database Systems. 1998:251-285.

[33] Haerder T, Reuter A. Principles of transaction-oriented database recovery[J]. Acm Computing Surveys, 1983, 15(4):287-317.

[34] Chawathe S S, Garciamolina H. Meaningful change detection in structured data[J]. Acm Sigmod Record, 1997, 26(2):26-37.

[35] Luo J, Deng W B. Transaction Control Based on Timestamp in Lazy Replication DDBS[J]. Computer Engineering, 2009, 35(23):73-75.

[36] 盖九宇, 张忠能, 肖鹤. 分布式数据库数据复制技术的分析与应用[J]. 计算机应用与软件, 2005, 22(7):36-38.

[37] Ge W. Research on the Application of Distributed Database Based on Oracle Advanced Replication[J]. Computer Engineering & Applications, 2003.

[38] Garmany J, Freeman R G. Oracle Replication: Snapshot, Multi-master and Materialized Views Scripts[M]// Oracle Replication: Snapshot, Multi-master & Materialized Views Scripts (Oracle In-Focus). Rampant TechPress, 2003.

[39] Daniels D, Doo L B, Downing A, et al. Oracle's symmetric replication technology and implications for application design[J]. Acm Sigmod Record, 1994, 23(23):467.

[40] Filip I, Vasar C, Robu R. Considerations about an Oracle database multi-master replication[C]// International Symposium on Applied Computational Intelligence and Informatics. IEEE, 2009:147-152.

[41] Paul S. Pro SQL Server 2008 Replication[M]. Apress, 2009.

[42] Gorelik A, Wang Y, Deppe M. Sybase replication server[J]. Acm Sigmod Record, 1994, 23(2):469.

[43] Song X B. A building of distributed database system based on sybase replication server[J]. Shandongence, 2000.

[44] Schwartz B, Zaitsev P, Tkachenko V. High Performance MySQL: Optimization, Backups, and Replication[J]. 2012(11).

[45] Bradford R, Schneider C. Effective MySQL Replication Techniques in Depth[J]. 2006.

[46] Chawathe S S, Garcia-Molina H. Meaningful change detection in structured data[J]. Acm Sigmod Record, 1997, 26(2):26-37.

[47] Chawathe S S, Rajaraman A, Garciamolina H, et al. Change detection in hierarchically structured information[C]// ACM SIGMOD International Conference on Management of Data. ACM, 1996:493-504.

[48] Lieuwen D F, Gehani N, Arlein R. The Ode active database: trigger semantics and implementation[C]// Twelfth International Conference on Data Engineering. IEEE Xplore, 1996:412-420.

[49] Labio W, Garcia-Molina H. Efficient Snapshot Differential Algorithms for Data

Warehousing[C]// International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. 1996:63-74.

[50] Shi J G, Bao Y B, Leng F L, et al. Study on Log-Based Change Data Capture and Handling Mechanism in Real-Time Data Warehouse[C]// International Conference on Computer Science and Software Engineering. IEEE, 2008:478-481.

致谢

时光荏苒，岁月如梭，我的研究生生活马上就要结束了。三年的时间又短暂又漫长，其中充满了回忆，更多的是成长与收获。2017 年那个夏天，初来华师，所有的场景依然历历在目。我非常庆幸自己能来到一个这么好的环境，而且还遇到了一群很好的老师和同学。由于自身基础比较薄弱，在学习和研究工作中遇到了很多困难，在老师和同学的热情帮助下，不仅顺利的解决了困难，同时也使我收获良多。在此，我要向所有关心我、指导我和帮助我的老师和同学们表示衷心的感谢。

首先，我要感谢我的导师宫学庆老师。研究生期间，宫学庆老师不仅在学术上指导我，同时在生活上也给予我很多的关心和帮助。在交通银行实习期间，宫老师严谨的工作态度和很强的工作能力令人钦佩，他渊博的专业知识也潜移默化的影响着我。此外，宫老师在工作之余还指导我完成科研论文，并且发表成功，我表示非常感激。

其次，我要感谢我的指导老师钱卫宁老师。钱老师对我的毕业论文从论文开题到论文撰写都提出了建设性的建议，使我收获很多。而且，钱老师每一次都非常认真和耐心的对我的论文进行修改，教导我论文写作的规范和标准，并给我举出具体的例子，让我更容易理解。此外，还要感谢周傲英老师的谆谆教诲。周老师的演讲有非常强的感染力，每一次都让人印象深刻。同时，我也要感谢实验室的张蓉老师、高明老师、张召老师和蔡鹏老师。他们对实验室的每一位同学都非常的用心，对有学术问题的同学都会倾囊相授。

然后，我要感谢实验室的郭进伟博士和王雷，在本论文的写作上他们给予了我很大的帮助。另外，还要感谢实验室的刘伯众、祝君、余晟隽、熊辉、储佳佳、李宇明、李捷莹等同学，大家学习和工作生活中都互相帮助、互相关心，每一个人都感受着温暖和快乐。还要感谢项目组的所有其他成员，大家一起学习、一起进步，使三年的研究生生活非常的充实。

最后，我要感谢我的父母和我的家人，他们无条件给予我关心和支持，默默

地鼓励我。没有他们，就没有今天的我，感激之情无以言表，唯有今后好好工作，努力报答他们。

谨以此文向所有关心和帮助过我的人们表示衷心的感谢。

发表论文和科研情况

■ 已发表或录用的论文

[1] 论文：分布式系统中 Semi-Join 算法的实现[J]. 华东师范大学学报(自然科学版), 2016(5):75-80. 第一作者

[2] 论文：OceanBase 数据库监控系统[J]. 中国数据库学术会议，计算机应用, 2016, 36(S1):237-239. 第二作者

■ 参与的科研课题

[1]国家自然科学基金，集群环境下基于内存的高性能数据管理与分析，2014-2018，参加人

[2]国家高技术研究发展计划（863 计划）课题，基于内存计算的数据库管理系统研究与开发，2015-2017，参加人