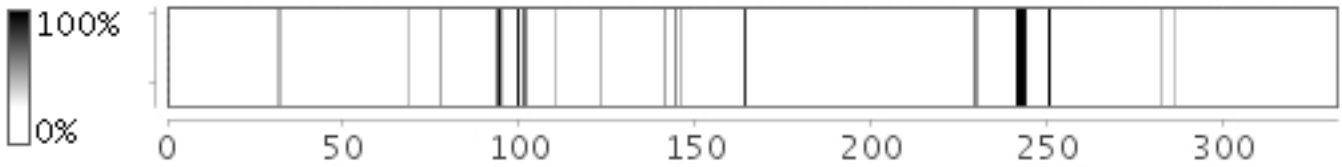
PaperPass旗舰版检测报告
简明打印版

比对结果（相似度）：
总　体：8 %　*（总体相似度是指本地库、互联网的综合比对结果）*
本地库：4 %　*（本地库相似度是指论文与学术期刊、学位论文、会议论文、图书数据库的比对结果）*
期刊库：3 %　*（期刊库相似度是指论文与学术期刊库的比对结果）*
学位库：2 %　*（学位库相似度是指论文与学位论文库的比对结果）*
会议库： 0 %　*（会议库相似度是指论文与会议论文库的比对结果）*
图书库：1 %　*（图书库相似度是指论文与图书库的比对结果）*
互联网：5 %　*（互联网相似度是指论文与互联网资源的比对结果）*

编　号：590EB843D4B1FLH69
版　本：旗舰版
标　题：基于Kafka 的分布式存储架构应用研究
作　者：吴瀚元
长　度：9864 字符(不计空格)
句子数：332句
时　间：2017-5-7 14:01:39
比对库：学术期刊、学位论文（硕博库）、会议论文、图书、互联网资源
查真伪：http://www.paperpass.com/check

句子相似度分布图：



本地库相似资源列表（学术期刊、学位论文、会议论文、图书）：
1. 相似度：1 % 篇名:《大数据下基于Spark的电商实时推荐系统的设计与实现》
　来源：学术期刊 《现代计算机》 2016年24期 作者: 岑凯伦 于红岩 杨腾霄
2. 相似度：1 % 篇名:《一种分布式消息队列研究与测试》
　来源：学术期刊 《物联网技术》 2016年8期 作者: 于金良 朱志祥 李聪颖

互联网相似资源列表：
1. 相似度：4 % 标题：《Kafka设计解析（五）- Kafka性能测试方法及Benchmar...》
　http://www.jasongj.com/2015/12/31/KafkaColumn5_kafka_benchmark/

2. 相似度：3 % 标题：《Hadoop Cluster Benchmark Performan...》
http://www.th7.cn/db/nosql/201612/219039.shtml
3. 相似度：1 % 标题：《SpringBoot入门Hello Wrod - toushita的...》
http://scholar.google.com/schhp?hl=zh-CN&as_sdt=40244b7bfe116a4c49852759a6eb7b0d

全文简明报告：

目 录

基于Kafka 的分布式存储架构应用研究

吴瀚元

摘 要： 在大型电子商务网站业务量飞速增长的今天，传统的数据存储架构愈发的无法满足新时代电子商务网站在响应速度、可扩展性、容灾性等方面的需求。 人们需要一种低成本的、可扩展的、高性能的大规模存储解决方案。 {42％：数据库的种类众多，而目前应用较多的两种是key-value数据库与关系型数据库，} {50％：而key-value数据库中Redis的使用量比较大，关系型数据库中Oracle的地位也很高。} 本文主要是对大型存储的需求进行分析，针对电商网站的的特点，同时使用两种数据库并使用Kafka作为中间件来同步数据， 既利用了Redis数据库易扩展高效率的特点，又利用传统数据库稳定性。 这一改进，不但提高了电商网站的访问效率，而且还能够方便的与客户已有的IT系统对接。 测试结果表明，本数据库存储架构的性能能够满足业务需求，与传统方式对比，效率大大提升。

关键字： kakfa； Redis； Oracle； 数据库

第一章 绪论

1.1 选题背景及意义

随着电子商务网站的快速发展，客户量与业务量的急速增长给数据库提出了更多的要求，而传统的数据库架构存在体积臃肿、扩展性不足、效率低下等问题， {40％：而新型的key-value数据库在较小数据量时性能虽然高，但是在大量数据高性能读写的应用场景中有硬件设备内存大小的限制。} 而且，一个大型的独立品牌电商网站在后台必然会与其原有IT系统的对接需求，单独某种单一数据库结构难以满足需求。

本项目是汉得信息技术股份有限公司的优衣库官网后台项目中的一部分。 在项目中有两种数据库同步的现实需求，既然单独应用两种数据库都无法解决问题，那么将两者结合在一起就成为了我们的选择。 而要将两者结合，其中最主要解决的问题就是数据同步。 为了保证数据同步的实时性和扩展能力，我们需要一种可水平扩展和高效率的消息系统作为数据转发的中转站。 目前有两种相对流行技术可以选择： Kafka与ActiveMQ。

通过对比，我们发现Kafka在吞吐量上有传统消息中间件不可比拟的优势。 而客户恰恰面对的式上亿级的用户。 而且在“双十一”购物节等线上购物活动期间并发的数据量极大，在吞吐量上有非常大的需求。 同时

Kafka使用了zookeeper技术作为集群支持，扩展性良好。

在本项目中，我们将Redis中的数据变化发送给Kafka集群，再由一个程序接收Kafka消息队列并将数据写入另一个Redis数据库。 同时有另一个程序负责将数据写入Oracle，从而解决了Redis集群与Oracle的数据同步问题。

本项目具有很高的现实意义，本系统已经在优衣库官网电商平台项目上得到实施，作为项目中的数据库架构，给前后端服务提供了稳定的、可扩展的支持。 本架构弥补了行业中Oracle数据库与Redis数据库同步的技术空白，为以后相关架构的发展提供了可靠的依据。

从个人学习进步的角度看，本选题题材比较前沿，对个人技术的发展和进步很有帮助，在开发过程中进一步熟悉了本架构中使用的各种组件， 在 Redis、 Kafka相关技术领域积累了很多的经验教训。

1.2 技术背景

本项目中使用的技术主要有： Redis、Oracle数据库、Kafka和spring-boot。 下面我们分别简要介绍这几种技术。

1.2.1 Redis

本项目使用Redis 数据库。 {45 %： 作为一个 key- value存储系统，它支持的数据结构有很多种类型，比如 bitmaps，散列（ hashes），} 列表（ lists），集合（ sets），字符串（ strings），有序集合（ sorted sets）与范围查询， hyperloglogs和地理空间（ geospatial）索引半径查询[1]。

Redis 有非常好的可扩展性，在3.0版本之后Redis-cluster计划被推出。 它支持节点的自动发现、选举、在线分片等机制。 {40 %： 而且集群的管理基于配置文件，用户只需要对配置文件进行简单的修改，再重启Redis即可完成集群的配置。} 非常方便。

本项目使用的Oracle数据库是Oracle Database 11XE版本。 在数据库领域中，Oracle Database的份额一直很高。{51 %： 作为一款关系型数据库产品，Oracle Database非常流行。} 它在多种操作系统平台上都可以运行，包括windows、linux、unix等。 同时也提供了多种语言的编程接口。 官方提供的管理软件SQLdeveloper也称得上强大易用[2]。

1.2.2 Kafka

本项目使用Kafka作为数据中间件。 Kafka是一个分布式的、分区的、复制的日志提交系统[5]。 在LinkedIn中它被用于处理活动流数据和运营数据。 在传统的应用场景中，Kafka也常常被用于实现集中式日志管理系统中的消息队列。

图1-1 Kafka应用场景

LinkedIn公司使用scala编写了Kafka。 后来才成为了Apache的项目。 如图1-1所示，Kafka通常被用作构建实时的数据通道，它是水平可扩展的，分布式的，快速的[6]。

1.2.3 Spring Boot

本项目使用Spring Boot 作为编码开发的Java框架，版本使用较新的1.4.0版本。 Spring Boot项目在Spring社区中非常年轻。 {64％： Spring Boot旨在帮助开发者更容易的部署Spring开发环境，更简单的创造基于Spring的服务和应用，让Java开发也能够实现Ruby on Rails那样的生产效率[4]。} {100％：为Spring生态系统提供了一种固定的、约定优于配置风格的框架。} Spring Boot具有如下特性[8]：

1.为基于Spring的开发提供更快的入门体验

2.开箱即用，没有代码生成，也无需XML配置。 同时也可以修改默认值来满足特定的需求。

{95％： 3.提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康}

4. {100％： Spring Boot并不是不对Spring功能上的增强，而是提供了一种快速使用Spring的方式。}

5. spring-boot的应用大大减少了开发中的配置任务，加快了开发进程[9]。

对于本项目而言，因为程序需要同时连接Redis与Oracle数据库，简洁的配置尤为重要，Spring Boot对关系型数据库与Redis数据库都有良好的支持。 对于Oracle来说，有spring-data-jpa； 对于Redis来说，有spring-data-Redis。

另外，本项目使用maven作为项目构建工具，使用github作为版本控制工具。

1.3 本章小结

本章介绍了开发本数据库架构的背景及意义，同时也指出了开发中要解决的核心问题是数据同步。 {46％：简要介绍了本项目中使用到的主要技术及其技术背景。} 结合项目中对技术的实际使用，在技术介绍中分析了使用相关技术路线的优势。

第二章 分布式架构的部署

2.1 整体结构说明

{50％： 图2-1 优衣库项目数据同步数据流向图}

项目的整体结构如图2-1所示。 本项目中涉及分布式部署的模块有Kafka与Redis两种。 为了降低组建之间的耦合度提高稳定性和扩展性，项目中使用了2个Kafka集群作为数据中间件，负责数据的转发。 MirrorMaker是一个Kafka的组件，能够实现两个Kafka集群中的数据复制。

左右两侧两个Redis数据库集群，代表应用环境中两个不同地理位置的数据中心，而下方的Oracle数据库则作为备份数据库使用。

这就是本项目目前的整体结构，为了看起来清晰，图2-1中省略了从右侧Redis2数据库到左侧Redis1的数据流向箭头。 至于从Oracle到Redis1的数据同步，由于项目需求等原因，本项目不涉及。

2.1.2 硬件部署

{54%：开发环境使用8台运行CentOS 6.4的服务器。} 服务器地址分别为172.20.1.50、172.20.1.51、172.20.1.52、172.20.1.53、172.20.1.54、172. 20.1.55、172.20.1.56、172.20.1.57。

配置如下：

1. CPU： 8 vCPU， Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz， 2 Sockets， 4 Cores per socket， 1 Thread per core

2.内存： 16 GB

3.磁盘： 500 GB

其中172.20.1.50~172.20.1.52这三台服务器上部署由三个节点组成的 Redis1集群（每台服务器一个 Redis节点），172.20.1.53~172.20.1.55这三台服务器上部署由三个节点组成的 Redis2集群（每台服务器一个 Redis节点）。 Oracle数据库部署在172.20.1.56上。 Kafka及其相关组件部署在172.20.1.57上。

2.2 Kafka集群

Kafka的架构从设计之初就是为分布式服务的。 {52%：在构成Kafka消息队列的组件中，生产者（producer）、消费者（consumer）、节点（Kafka cluster）都可以有多个。}

图2-2 Kafka生产消费关系图

如图2-2所示，消息（在本项目中即用户操作数据库的增量数据）由生产者（producer）发出，再由broker进行分发给所有的消费者（consumer）。 {63%：而生产者（producer）和消费者（consumer）的注册则由broker作为接口完成。} broker同时也作为活动数据与消息处理系统之间的缓存代理。 {61%：客户端与服务端之间使用TCP协议进行通讯。} 使用者甚至无需理会Kafka内部的通讯方式，只需要调用API即可实现大多数功能。

下面介绍几个Kafka的重要名词：

1. Topic： 可以理解为Message（消息）的分类，不同分类的消息分别由不同的Producer和Consumer生产和消费。

2. Message： 消息，是通信的基本单位，即Producer生产的内容。

3. Producers： 消息和数据生产者，由它发送message到对应的topic中，在本项目中producer由Kafka的Java api实现。

4. Consumers： 消费者，订阅并消费指定topic的message。 在本项目中consumer由Kafka的Java api实现。

5. Broker： {87%：缓存代理，Kafka集群中的一台或多台服务器统称为broker[6]。}

在本项目中，每个kakfa集群有由3个Kafka-server与一个zookeeper（分布式应用协调服务）组成。 每个数据流动方向使用一个独立的Topic，比如从Redis1到Redis2的数据流使用一个叫test1的topic。 Producers与Consumers则通过Kafka提供的java的api来实现。 发送的Message即为数据同步所需的数据变化信息（主要包含add、update、delete三种操作类型及其数据）。

2.3 Redis集群

Redis集群部署比较简单。 修改Redis.conf中关于数据库集群节点以及IP地址与端口的相关配置，重新启动Redis数据库即可。 {53%：在应用环境中，每个Redis集群由三台Redis服务器节点组成。} 而在生产环境中，受限于设备硬件性能，我们使用单节点的伪集群部署。

图2-3 Redis集群部署示意图

Redis集群有如下特征：

(1)每个Redis cluster之间互相连接，通过配置，既可以分别存储不同的数据，也可以互为备份。 节点之间的通讯采用二进制协议。

(2)客户端与cluster直接连接，即客户端无需与所有cluster都建立连接，与任意可用cluster连接之后都可以对整个集群进行交互

(3)选举机制，由cluster自发选举出master节点，目前版本需要奇数个数的节点来保证不出现“脑裂”现象。

2.4 本章小结

本章主要介绍了项目在架构上的整体设计，着重介绍了kakfa集群的几个基本概念。 同时分别描述了Kafka与Redis在本项目中的部署情况。 环境的部署为整个项目的运作提供了基础支持，而项目的可扩展性也依赖于这些组件本身。 其中，Kafka的扩展能力尤为重要，我们期望它的吞吐量能够随着模块的增加线性增长，这一点会在第四章得到验证。

第三章 数据同步

3.1 数据

解决数据同步问题要先考虑数据是如何产生的。 对于本项目的应用背景（一个电商平台）而言，数据操作无非是add、update和delete的组合。 所以，只要能收集到操作类型、操作的目标（Oracle中的表或者Redis中的HashValue）、操作的数据（每个字段以及对应的值），我们就可以将数据原封不动的“同步”到另一个数据库中。 在这里同步的本质不是复制，而是将数据操作原样执行了一遍。 这就是本项目中数据同步的基本原理。

在实现中，用户在对Redis1进行操作的时候，数据的增量信息（操作类型、操作的目标、数据）就会插入到Redis中的一个队列（从左向右，先进先出）里。 随后再由程序从右侧弹出队列，并将信息发送到Kafka集群中。

3.2 消息的顺序

对于消息的顺序而言，Kafka并不是一个完全有序的消息队列。 Kafka只保证在一个group中只有一个consumer的情况下，topic的是有序的。 但是在应用情况中，为了提高性能保证速度，producer与consumer都是多线程的，也就是说，同时有多个producer在生产数据，也同时有多个consumer在消费数据。 这就会导致数据的无序。 即，在本项目中add、updtae、delete数据的操作的先后顺序很可能是混乱的。

对于数据库来说，add、updtae、delete数据的操作顺序的改变很明显会导致最终结果的不同，所以我们需要有一种机制来保证即使数据操作的顺序不同，最终结果也是一样的。

因此这里我们引入了“伪删除”机制，即删除操作是将数据从原表删除的同时存入另外一张我们称之为“删除表”的表中。 同时我们给每个操作一个ID作为唯一标识。 然后通过修改add、delete、update的操作逻辑实现这一功能。

下面分别介绍三种操作类型的流程：

add：

图3-1 add操作流程图

{41%：如图3-1所示，在进行 add操作时，先判断“删除表”中是否有同 ID数据，} 如果有则不进行操作（这意味着本条在原始操作中顺序早于 delete的 add操作晚于 delete操作的消息到来）， 否则，判断“普通表”是否有同 ID数据（为了区分由于某些意外原因同样的数据被发送了两遍）， 如果有同 ID数据则不进行操作，否则执行 add操作（一切正常）。

update：

图3-2 update操作流程图

如图3-2所示，在进行 update操作时，先判断“删除表”中是否有同 ID数据， 如果有则不进行操作（这意味着本条在原始操作中顺序早于 delete的 update操作晚于 delete操作的消息到来）， 否则判断“普通表”中是否有同 ID数据，如果无则将 update操作转为 add操作（这意味着本条在原始操作中顺序晚于 add的 update操作早于 add操作的消息到来）， 如果有则判断消息的版本号，大于则执行，小于则不执行。

delete：

图3-3 delete操作流程图

如图3-3所示，在进行 delete操作时，先判断“普通表”中是否有同 ID数据，如果无， 直接将删除内容插入“删除表”中，如果有则执行 delete操作，再将删除内容插入“删除表”中。

通过上述三种数据操作类型的逻辑设计，我们巧妙的做到了即使消息到来的先后顺序不对，当数据操作完成时的数据结果完全相同。 解决了数据同步中Kafka消息的无序问题。

3.3 本章小结

因为数据同步是本架构要解决的最核心问题，所以本章主要介绍了数据同步的基本原理，着重介绍了Kafka无序问题的解决。 通过引入删除表并修改增删改操作的内部逻辑，这个问题的解决打通了多个数据中心之间数据同步的通道。 而这一通道的打通，意味着跨数据中心的数据同步功能实现了。

第四章性能与测试

4.1 测试环境

{91%：本文的测试共使用6台安装CentOS 6.5的虚拟机，3台作为Broker，另外3台作为Producer或者Consumer。} 每台虚拟机配置如下：

4. CPU： 8 vCPU， Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz，2 Sockets，4 Cores per socket，1 Thread per core

5.内存： 16 GB

6.磁盘： 500 GB

4.2 测试方法

Kafka官方提供了多种测试脚本，我们使用其中的两个进行测试：

1.$ Kafka_ HOME/ bin/ Kafka- producer- perf- test. sh该脚本被设计用于测试 Kafka Producer的性能， {100%：主要输出4项指标，总共发送消息量（以 MB为单位），每秒发送消息量（ MB/ second），} {100%：发送消息总数，每秒发送消息数（ records/ second）。} {99%：除了将测试结果输出到标准输出外，该脚本还提供CSV Reporter，即将结果以CSV文件的形式存储，便于在其它分析工具中使用该测试结果。}

2.$Kafka_HOME/bin/Kafka-consumer-perf-test.sh 该脚本用于测试Kafka Consumer的性能，测试指标与Producer性能测试脚本一样。

通过配置这两个脚本不同的参数（可手动调整数据包大小、发送时长等），我们获得了几组测试信息，将数据整理成图表之后我们发现测试结果符合预期。

4.3 测试结果

测试一： producer数量与总吞吐量线性相关测试

测试目标： {100%：多个Producer可同时向同一个Topic发送数据，在Broker负载饱和前，理论上Producer数量越多，集群每秒收到的消息量越大，并且呈线性增涨。}

测试结果：

图4-1 Kafka生产者数量与吞吐量关系图

如表4-1所示，测试结果满足期望。 即当我们多线程使用Kafka生产者发消息时，Kafka的吞吐量线性的增长。

测试二： 消息体积大小与吞吐量关系

测试目标： 找出合适的消息长度，在单条消息长度满足使用需要的同时，能够维持一定的数据吞吐速度（MB／s）， 理论上来说，消息越短，每秒消息吞吐量（records／s）越大， 数据吞吐速度（MB／s）越小。在实际项目中，message的大小会根据业务需求调整，我们通过这个测试，可以作为未来判断带宽需求的依据。

测试结果：

图4-2 Kafka消息体积与吞吐量关系图

如表4-2所示，在应用中数据吞吐速度（MB／s）越快，在带宽固定的情况下的延迟越低， 所以可以根据图上的节点根据不同业务类型的需要来选择合适的数据包长度。

4.2 本章小结

在前面的三章中，我们对项目背景、技术路线、项目中的核心问题等关键信息做了介绍。 在经历了3个月左右时间的开发之后，整个项目开发完成，已经应用到了客户环境中，运行情况良好。

作为应用研究，测试必不可少。 通过测试，既可以完整的测试出本项目各个模块的性能，为后续的扩展开发作为参考，又可以与传统架构做对比，以显示出本架构的优越性。

结论（总结及展望）

此次应用研究中，主要是基于电商平台的实际项目，应用Redis、Oracle两种数据库并使用Kafka作为中间件的一个数据架构解决方案。 在应用研究中，我们主要解决了数据同步这一关键问题，解决了增量数据的获取方法，也解决了Kafka在多个producer与consumer时的无序问题。 测试的结果也显示，解决了无序问题之后，Kafka的扩展性得到了充分的利用，数据吞吐量随着模块数量的增多线性的增长。

作为本次应用研究而言，Redis-Oracle-Kafka数据库架构的“主体枝干”基本完成，但依旧保留了一些可供扩展的地方： 比如Oracle数据还原到Redis的过程，比如缺少一个可视化的监控工具等等。

在研究过程中也走过一些弯路，比如在增量数据获取的方案中，最初使用Redis自带的广播机制，忽略了增量数据的存储问题（广播消息不保存）。 同时这些尝试也带给我很大的收获。 虽然Redis广播机制不适合作为数据获取的工具，但是在未来开发的开发中Redis广播可以用来监控数据库的运行状况。

最后，希望本数据库架构能够在更多的地方得到应用，也希望更多人能够从本文中得到启发。

致谢

{43％：在这次毕业设计从定题到完成的整个过程中，有很多人给予我很大的帮助，我在此感谢他们。}

首先要感谢我的指导老师，刘春花老师从论文的选题到论文的写作等毕业设计各个阶段中都提供了很多中肯的意见，指点迷津、指引方向。

其次要感谢我的同事，他们在工作之余给了我很多技术方面的指导，包含技术路线的选择这样的大方向与一些编程语言的基础知识等。

还要感谢的就是我的公司，他们提供了强大的服务器与网络资源，这些是我完成研究的硬件基础。 {42%：当然还要感谢各个开源社区，没有他们就没有kakfa，也就没有我完成本次应用研究的技术前提。}

谢谢你们！

参考文献

[1] Josiah L. Carlson. 《Redis IN ACTON》[M]. 人民邮电出版社.2015

[2] Bob Briyla, Kevin Loney. 《Oracle Database 12c: The Complete Reference（第7版）》[M]. 清华大学出版社.2015

[3] Redislab. The Redis Documentation. https: //Redis.io/documentation.2002

[4] Phillip Webb, Dave Syer, Josh Long, St é phane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, S é bastien Deleuze. Spring Boot Reference Guide. http: //docs.spring.io/spring-boot/docs/1.4.6.RELEASE/reference/htmlsingle/ .2017

[5] A distributed streaming platform. http: //Kafka.apache.org .2017

[6] Kafka 0.10.2 Documentation. http: //Kafka.apache.org/documentation/ .2017

[7] Jason. Kafka性能测试方法及Benchmark报告.

http: //www.jasongj.com/2015/12/31/KafkaColumn5_Kafka_benchmark/ .2017

[8] 汪云飞.《JavaEE开发的颠覆者： Spring Boot实战》.电子工业出版社.2016

[9] 陈韶健.《深入浅出Spring Boot》.机械工业出版社.2016

[10] 黄健宏.《Redis设计与实现》. 机械工业出版社.2014

[11] 丁士峰.《Oracle PL/SQL从入门到精通》.清华大学出版社 .2012

[12] Nishant Garg. 《Apache Kafka》. PACKT open source.2016

[13] 皮雄军.《NoSQL数据库技术实战》.清华大学出版社 . 2015

[14] Sourabh，Sharma. 《Mastering Micoservices with Java》. 电子工业出版社.2017

[15]胡谦.《基于Kafka的日志实时分析系统的设计与实现》.论文联合库.2015

[16]Neha Narkhede，Gwen Shapira，Todd Palino.《Kafka: The Definitive Guide》 [M].O ’Reilly Media.2017

[17]李林峰.《分布式服务框架原理与实践》[M].电子工业出版社.2016

[18]Flavio Junqueira.《ZooKeeper: 分布式过程协同技术详解》[M].机械工业出版社.2016

[19]Sam Newman.《微服务设计 (图灵程序设计丛书)》[M].人民邮电出版社.2016

[20]李子骅.《Redis入门指南》[M].人民邮电出版社.2015