各位答辩老师，同学大家好，我是2010级03班专硕瞿龙俊，我的指导老师是李星毅老师，今天我的答辩题目是基于HBase的交通流数据实时存储与查询方案的设计与实现。**【翻页】**

本次答辩我将从五个方面分别进行阐述，首先我将简单介绍论文的相关研究背景与内容，然后，我会详细介绍一下基于该优化方案的整体架构设计思路，接着，从数据存储和数据查询两个方面对关键环节相关设计进行阐述，最后，总结论文所作出的相关贡献和进一步改进之处。**【翻页】**

首先，我先简单汇报一下论文的相关背景、研究现状与内容创新点。**【翻页】**

本人论文选题主要受四个因素的影响，首先，【因素一】【灰色框】，这是客观因素，其次，【因素二】，这是需求因素，接着，【因素三】，这是技术支持因素，最后，【因素四】，这是现状因素，基于上述因素的考虑，我的研究改进了现有的数据方案，提出了一种基于HBase的交通流数据存储与查询优化方案。**【翻页】**

在对HBase的存储与查询优化方面，国内外有较多研究成果，首先，华为公司研发了一个非主键索引的查询系统Hindex，它采用基于Region的局部索引模型，通过对HBase表的每个Region建立单独的索引表来提高非主键查询的效率，但它的缺点是，绝大多数检索任务的目标记录数量相对较少，在分布式集群中并行执行该任务会造成很多未存储任何目标的节点也触发了检索过程并返回空集，会消耗不必要的计算资源，降低并发查询吞吐量和查询效率。，

然后，NGDATA公司的产品HBase-Indexer也提供了一种Hbase上的非主键索引，它将更新数据异步发送到索引服务器集群，在索引服务器上分析数据并生成对应的索引数据，查询数据时先访问索引服务器来定位HBase的内容，这种机制采用异步更新，时效性较差。希腊帕特雷大学提出了一种非主键范围查询的索引方案，它通过MapReduce构建检索线段树并保持在内存中，并将每段范围的上界保存在HBase表中，该方案扩展性好，但对于单点查询，线段树退化成二叉树，索引空间开销和维护代价都很高。**【翻页】**

我在现有前人的研究成果基础上，结合交通流数据的特点，提出了一种基于HBase的交通流数据存储和查询的优化方案，主要研究内容有三部分，首先，【内容1】，然后，针对数据存储和数据查询两个方面，分别进行了相关优化的设计，在数据存储方面，【内容2】，在数据查询方面，【内容3】。 我研究的主要创新点主要集中在数据存储与查询优化的设计上，在数据存储方面，首先提出了Hbase行键存储模型，然后针对数据存储时产生的相关问题，提出了预分区数据分片策略和存储调度算法，在数据查询方面，提出了Redis缓存记录存储模型和基于热度积累的缓存淘汰算法。**【翻页】**

首先，我先介绍一下基于HBase的交通流数据存储与查询优化方案的整体架构设计。**【翻页】**

它主要分为四个模块，前端模块，写入缓冲，缓存控制和HBase集群。**【翻页】**

如图所示【点击】，它可以分为左右两个部分，左侧部分是数据存储的优化方案，右侧部分是数据查询的优化方案【点击】。前端识别模块主要是【前端模块介绍】【点击】，写入缓冲模块主要提供存储时的相关数据处理，主要包含数据预处理层、缓冲层和数据写入模块【点击】。缓存控制模块主要提供查询时相关优化设计，在缓存控制模块，提供了多级缓存设计方案。利用本地缓存和Redis集群缓存，提供了高效的数据查询方案，在缓存控制模块，利用缓存淘汰算法对缓存容量进行控制，使缓存控制能够在合理范围内运行【点击】。Hbase集群是数据持久化的解决方案，负责存储所有数据和处理查询请求

下面我针对数据存储相关关键技术进行详细阐述**【翻页】**。

我的研究以交通流数据中较为常见的道路车辆信息数据为例，首先介绍一下道路车辆信息数据模型，它主要有如下几个数据字段构成：【介绍字段】。在这些数据字段中，最关键的字段是车辆车牌记录、监测点编码，和车辆识别时间，因此，我们将这三个字段构成核心数据，其他若干字段构成非核心数据，在设计HBase主键结构时，应充分考虑车辆车牌记录、车辆识别时间、监测点编码这三者之间的关系，将它们作为复合主键使用。**【翻页】**

合理的HBase主键模型设计不但可以提升检索效率，降低冗余带来的性能问题，同时一定程度上也能解决“写热点”问题。根据HBase数据检索策略，将数据拆分到不同的Region可以提升并行查询的能力，在主键设置标识前缀，依据Region拆分合并策略，HBase会将记录保存至指定Region。我提出了一种HBase的主键生成策略，它由标识前缀（PREFIX）、车辆车牌信息（CAR\_NO）、监测点（POINT\_NO）和记录时间（RECORD\_TIME）四个属性复合而成。 如示例所示，它表示查询一条车辆车牌记录为沪B37D83，于2017-03-05 00:01:33在编号为10300501的监测点记录，该条记录被保存在相应的Region中。本策略使用了前缀，避免了读写热点，使得数据分布更均匀，更适合交通流数据存储业务场景，能充分利用HBase集群的计算和存储性能。 **【翻页】**

下面我来讲一下整个数据的存储流程，存储一条记录需要经过四个步骤，分别是数据预处理层、缓存处理器、缓冲队列和数据写入层，数据预处理层主要对数据进行清洗和整理操作，缓冲处理器对数据进行处理，形成适合写入的指令进入相应的分类缓冲队列进行等待，然后进入数据写入层，按照指定规则写入HBase集群中。下面我详细介绍各个步骤。**【翻页】**

在数据预处理层，主要对数据进行三个操作，分别是【三个操作】。标记分类是对数据进行标记数据源，例如【解释点1】等等。脏值处理是指对部分数据进行清洗操作，由于数据采集依靠硬件设备，无法完全保证数据采集的有效性，因此需要对数据进行校验，如果某条记录的核心数据丢失了，则该条记录的存储将没有意义，因此直接过滤该条数据的存储，对于非核心数据的缺失，可以采用相关备用方案进行数据补全操作。规范格式是指对数据进行格式化，将不同来源不同格式的数据按照指定的属性排列顺序进行格式化，方便后续流程进行处理。在进行完成上述操作之后，数据会进入缓冲层进行处理。**【翻页】**

数据缓冲层接收到来自预处理层的数据之后，会对数据进行深加工，主要有缓冲接收处理器、缓冲队列控制和等待队列构成。缓冲接收处理器的主要任务是，将之前传入的字符串形式的数据进行序列化，根据预设的规则进行拆分

缓冲队列控制主要有两个功能，第一，用来对等待队列进行管理，动态的调整等待队列的数量与阈值，第二，将上一步接收到的数据根据数据源的不同，将数据放入不同的等待队列中。等待队列通常由消息队列实现，由生产者在队头写入操作，消费者在队尾读取数据操作**【翻页】**。

这是一个数据记录序列化的实例，每一条记录值，按照表格上方的解析规则进行拆分，拆分结果如表格右侧的序列对象所示的，将字符串类型的记录值解析为序列Object。

缓冲队列有多种开源实现方式，但大部分实现方式均遵循消息服务的标准或规范，目前主流的实现方式有两种，一种是P2P的实现方式，涉及到三个概念，消息队列、发送者和接受者，每个消息都被发送到一个特定的队列，接收者从队列中获取消息。队列保留着消息，直到他们被消费或超时。另一种是发布订阅者模式，即由主题、发布者、订阅者三个部分组成，客户端将消息发送到主题。多个发布者将消息发送到Topic,系统将这些消息传递给多个订阅者。具体使用哪种模型取决于交通流数据的类型，如果希望发送的每个消息都应该被成功处理的话，那么需要P2P模式。如果希望发送的消息可以不被做任何处理、或者被一个消息者处理、或者可以被多个消费者处理的话，那么可以采用发布-订阅者模型。**【翻页】**

当数据缓冲层将数据处理完成之后，数据就会通过等待队列进入数据写入层，数据会在写入层写入HBase集群。

HBase内置了一种处理拆分Region策略。它的拆分策略是这样的：每一个Region使用属性Start Key和End Key进行数据存储区间的划分。默认情况下，HBase在初始化阶段只会创建唯一Region，随着数据量的增加，一旦它们达到了阈值，将会触发HBase的拆分操作，将当前Region通过Middle Key拆分为两个Region，新的Region可以接受数据并继续增长。对于交通流这类流量较大，数据复杂，查询与存储请求频繁的应用场景，使用这种方案会导致性能问题。首先，使用内置拆分策略会导致“拆分风暴”，即当Region的大小以恒定速率保持增长时，Region会在同时进行拆分操作，因为拆分需要压缩当前Region中的存储文件，重写拆分之后的Region，会导致I/O延迟的上升。其次，在进行Region拆分时，旧的Region会离线，此时部分数据会因为拆分操作导致无法访问，形成存储热点，无法满足实时性要求较高的应用场景。**完备性**：每条逻辑记录t均会被映射至对应逻辑片段，即每一条数据记录，均会在数据库中找到唯一对应的区域位置。 **可重构性**：存在逻辑片段Si可以被看作全局关系R进行再次分片。**不可相交性**：对于唯一逻辑记录t，会且仅会映射至唯一逻辑片段Si上。在存储数据时，对于不同数据进行根据社群特征分类，为不同类别创建对应的Region存储记录。例如，若为市级交通流数据存储方案，则可以考虑使用监测点编号（ROAD\_NO）进行社群分类，若为全国交通流数据存储方案，则可以考虑使用省市作为分类标准。**【翻页】**

**【翻页】**

在前面研究的基础上，我对数据写入模块部分功能进行了一些对比试验测试，数据来源为国内某市2015年06月至2015年10月期间各路口路段监测设备采集数据、车辆全球定位系统数据等。 这是数据写入性能对比实验的实验结果，分别比较关系型数据库MySQL、原生HBase集群和我研究的优化方案的在数据写入频率为2k、4k、6k…20k时的延迟时间。实验结果延迟时间越小，数据写入性能越好。 MySQL数据库响应延迟时间随着数据发送速率成倍增长。对于原生HBase集群数据库，在五千条每秒数据发送频率以下时效率较高，当每秒数据量超过五千条时，原生HBase性能下降较快。而对于优化方案，由于缓冲队列和并发写入的存在，响应延迟时间随着每秒发送速率变化的曲线较平缓，当每秒数据发送请求速率达到2万条时，才出现了较高的响应延迟时间。 **【翻页】**

这是缓冲队列写入阈值的测试实验，用来验证写入阈值对数据存储性能的影响。图中横坐标为数据发送速率，分别为2千条/秒、4千条/秒直至2万条/秒，纵坐标为响应延迟时间，单位为毫秒，当数据发送速率不断增加时，不同的缓冲队列写入阈值对性能影响较大。在数据发送速率低于一万条每秒时，不同的阈值对性能影响较小，当超过一万条每秒时，阈值为6M的曲线较缓，是理想的性能状态。低于6M的阈值由于I/O频繁导致性能降低，高于6M导致数据长时间滞留缓冲区，性能也会降低。 由此说明，6M是一个合适的阈值范围。**【翻页】**

下面，我针对数据查询的优化方案进行详细阐述。**【翻页】**

首先，我先介绍Redis集群缓存服务器的整体架构，如图所示，主要由客户端模块，KeepAlived模块、分布式代理中间件TwemProxy、监控模块组成。【点击】客户端模块，是指用户通过中间件向Redis集群发起读写命令。 【点击】客户端使用虚拟IP地址对Redis集群进行访问，KeepAlived提供虚拟IP地址和端口。可以将多个无状态的单点通过虚拟IP漂移的方式构建高可用服务。KeepAlived实现基础是虚拟路由冗余协议。【点击】TwemProxy是一款Redis单线程代理中间件，是缓存服务器集群管理工具，TwemProxy可以减少对Redis集群的连接数。通过TwemProxy可以将多台通用服务器水平扩张Redis，有效的避免单点故障问题且支持失败节点自动删除，TwemProxy支持一致性哈希，可以将客户端请求的数据通过计算分发到相应的Redis实例中。【点击】监控模块由Sentinel和Redis-Agent组成，负责监控主从服务器是否正常运行，自动故障转移服务和提供相关配置。**【翻页】**

缓存命中率是缓存设计的主要目的，将频繁访问的记录映射至Redis服务器，可以大幅提升记录查询效率。我研究并改进了一种缓存记录存储模型，下面我详细介绍一下各个字段的意思。【点击】Flag表示索引标识符，用来标识当前所属的HBase非主键的列属性别名。【点击】Record表示列字段记录值，是指HBase中当前索引列的具体数值。【点击】Index表示当前索引列中，当前记录值对应的Hbase主键信息的集合。【点击】Frequency表示在时间周期内，该索引列的记录值根据被访问频率和历史频率所计算出来的热度值。【点击】Count表示时间周期内，当前缓存记录值被查询请求访问的次数。**【翻页】**

当遇到数据检索瓶颈时，设置缓存是一种行之有效的方法，然而随着访问量的增加，数据总量也越大，命中率降低。因此，需要设置一个缓存淘汰算法，将部分访问频率低的记录淘汰出缓存中，常见的缓存淘汰算法有三种：【介绍三种算法】。介绍【先进先出算法】，介绍【最近最少使用淘汰算法】，介绍【最近最不常用淘汰算法】**【翻页】**

综合参考基于访问频率的淘汰算法和基于最近访问时间的淘汰算法，提出了一种基于热度积累的缓存淘汰算法，热度是指在单位时间内，当前记录值被访问到的次数，下面我介绍一下各个字段。【点击】SCORE表示计算出的热度值，SCOREN-1表示前N-1个时间周期内，根据访问频次计算出的热度值 。【点击】α为权重系数，用来平衡当前时间周期记录被访问的热度值和前N-1时间周期的热度值之间的影响力因子，α越大，则表示当前周期记录被访问的频度对SCORE影响越大，反之历史热度影响越大。 【点击】visitCount为访问次数 【点击】countPeriod为时间周期。visitCount/countPeriod表示时间周期内访问次数/时间周期，即当前时间周期记录被访问的热度值。**【翻页】**

在多级缓存中，本地缓存作为第一级缓存，提供相比Redis集群缓存更快速的数据查询服务，区别于Redis集群，本地缓存仅保存小数据量的数据，且该数据热度值较大。当客户端发起查询请求时，若配置开放了本地缓存，则会优先进入本地缓存检索。 我提供了两种本地缓存的实现策略，一种是LRU算法实现，一种是LFU算法实现，两种都基于LinkedList链表实现，数据均封装成Node对象，均提供新增、更新和删除接口，区别是LFU算法需进行二次封装，需要引入Count进行计数，在LRU算法实现中，利用的原理是【LRU原理】，而在LFU算法中，利用的原理是【LFU原理】。**【翻页】**

为了测试当前数据查询优化方案的可行性，我进行了数据读取性能的对比试验，分别比较使用缓存服务器和不使用缓存服务器分别在数据量不断扩大的情况下，数据写入性能的对比。随着数据量的增加，查询时间也相应延长。在数据量少于300万条时，直接进入HBase集群查询时间比使用缓存模块更少，原因是在数据量较少时，查询数据集较少，此时大量时间被用来扫描HBase集群并构建缓存，而不使用缓存模块则无需构建缓存，因此使用该方案的查询效率更高。随着数据量的变大，查询数据集的不断扩大，缓存模块的优势也逐渐体现，当数据量达到5亿条时，使用缓存模块的查询时间比不使用缓存模块节约了近20%，查询效率提升了25%。**【翻页】**

缓存索引命中率是考察一个缓存淘汰策略是否合理的重要指标，命中率越高，则说明缓存效率越高，我进行了一个对比实验，分别对比LRU算法和基于热度淘汰算法的缓存命中率，实验结论证明，缓存命中率与缓存容量具有线性相关性，缓存容量的扩大会引起缓存命中率的提升，当缓存容量达到总缓存空间的80%时，命中率达到最大**【翻页】**