**南方科技大学**

**硕士学位论文中期报告**

**题 目：**一种基于Spark的分布式混合索引结构的设计与实现

**院 （系） 计算机科学与工程**

**学 科 计算机科学与技术**

**导 师 Elvis Sze-yeung Liu**

**研 究 生 战庶**

**学 号 11849062**

**中期报告日期 2019年12月5日**

**研究生院制**

目 录

[1．课题主要研究内容及进度情况 1](#_Toc445315930)

[1.1．课题主要研究内容 1](#_Toc445315931)

[1.2．进度情况 1](#_Toc445315932)

[2．目前已完成的研究工作及结果 2](#_Toc445315933)

[2.1．系统仿真模型 2](#_Toc445315934)

[2.1.1系统仿真模型的建立 2](#_Toc445315935)

[2.1.2系统仿真模型的验证 3](#_Toc445315936)

[2.2匹配滤波 4](#_Toc445315937)

[2.3符号同步 5](#_Toc445315938)

[2.3.1 闭环Gardner算法 6](#_Toc445315939)

[2.3.2 开环非线性处理算法 10](#_Toc445315940)

[2.3.3 定时误差校正算法 14](#_Toc445315941)

[2.3.4 开环和闭环系统算法性能对比 16](#_Toc445315942)

[2.3.5 减少定时同步抖动的预滤波器设计 17](#_Toc445315943)

[2.4载波同步 19](#_Toc445315944)

[2.4.1 DFT频率粗估计算法 19](#_Toc445315945)

[2.4.2 维特比频率估计算法 23](#_Toc445315946)

[2.4.3 维特比相位估计算法 25](#_Toc445315947)

[2.5结论 26](#_Toc445315948)

[3．后期拟完成的研究工作及进度安排 27](#_Toc445315949)

[4．存在的困难与问题 27](#_Toc445315950)

[5．如期完成全部论文工作的可能性 27](#_Toc445315951)

1．课题主要研究内容及进度情况

1.1．课题主要研究内容

近年来，随着海量规模的大数据产生，对大数据处理技术的需求越来越迫切。作为时下非常流行的大数据处理框架Spark，主要研究的是如何构建一个分布式混和索引框架，从而提高数据检索速度。由此开发了一个分布式大数据索引系统，它是基于Spark框架，将创建的索引容器封装为RDD，这样会具有RDD的容错机制。 然后使用Spark Sql解析器执行查询操作，查询存储在数据仓库SQLite中的数据，并使用混合索引策略，提高查询性能。

在本课题中，我们提出了一个基于Spark的高效，可扩展，分布式的索引框架。我们的框架有很多的优点，首先，它通过优化的索引结构，大大缩减了搜索的时间。其次，有很强的扩展性，可以在此基础上继续进行开发。最后，将这个系统进行了封装，同时具有了潜在的商业价值。

1.2．进度情况

目前，课题已完成了分布式计算环境的搭建，从网上爬取数据作为数据源，并完成了混合索引的设计和算法的研究，并开发了web前端显示界面与用户信息交互。需要进一步完成的是，将混合索引方案与传统索引方案性能的比较，以及进行代码的测试和优化。

表1.1 研究进度



2．目前已完成的研究工作及结果

课题首先建立了分布式混合索引模型，然后索引系统实现中使用的算法部分进行了讨论。

**2.1 混合索引结构**

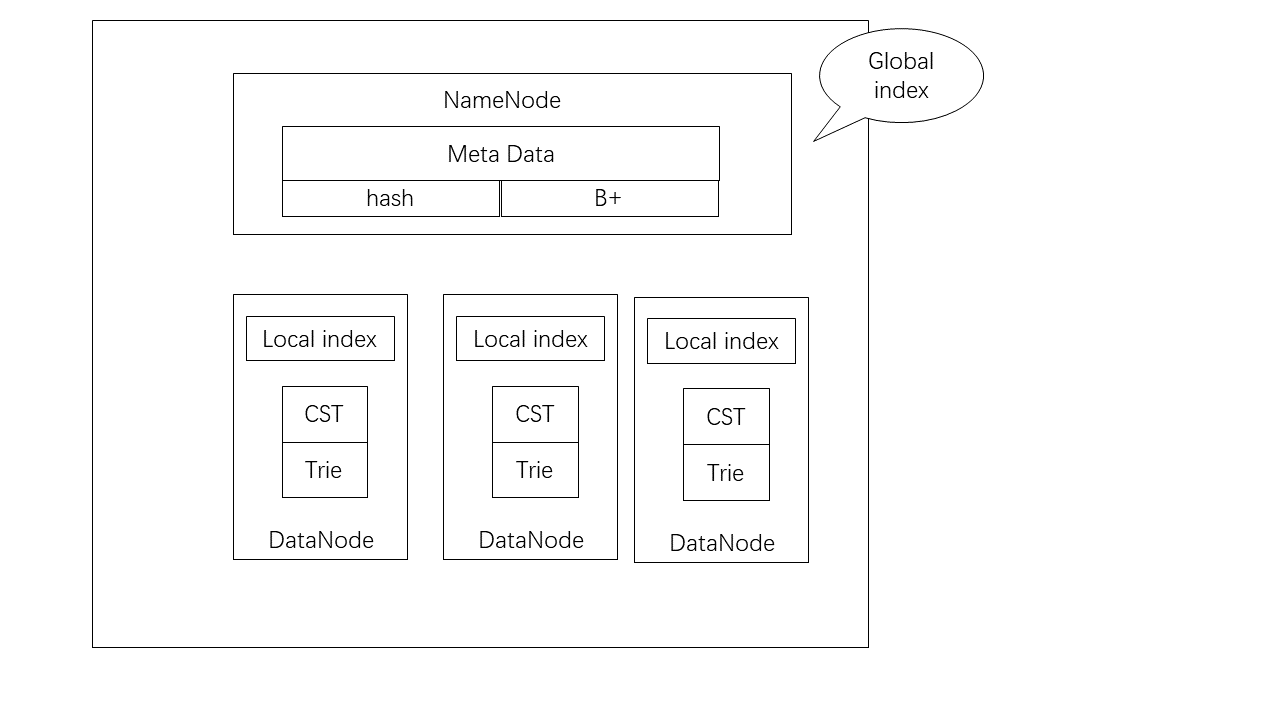
**2.1.1 混合索引结构设计思想**

索引整体采用分层分布式索引结构。分层分布式索引结构，首先将数据根据SQL分布进行划分，将临近的数据划分到一个分区中，再对每个分区中的数据构建一个局部索引，并将局部索引以及分区的数据存储到数据节点中。在所有局部索引构建完成后，在主节点根据局部索引的根节点构建一个全局索引。由于全局索引只保留局部索引根节点的信息，限制了全局索引的大小，降低主节点的运算规模。

混合索引策略的出发点是能够结合不同索引更新策略的长处，将不同索引更新策略混合，以使索引过程更高效。

全局索引：全局索引是对整个数据系统建立的统一索引，每个索引指向一个节点或者数据块。对于标签数据，使用hash索引，在查询时先根据全局索引访问相关的节点以及数据块即可，能有效减少数据读取的时间，提高检索效率。对于经常访问、命中率较高的数据，建立B+树索引进行检索。因为B+树具有搜索路径短的特点，适合进行频繁访问。

局部索引：局部索引是数据分区单独建立的索引，数据块之间检索是相互独立的，这样会提高容错能力，同时也有利于数据块更新和索引的重新建立。在局部索引结构中，根据数据的特点，采用不同的索引策略。采用CST树和Trie树索引相结合的方式。对于数值型数据，使用CST树，因为T树叶子节点具有有序性，保证了它在数据检索上更加高效。但是T树对于跨数据范围检索时，往往采取结点分裂的方式，这样重新调整树结构会花费大量时间。对于文本型数据，选择使用Trie树更优。因为Trie树可以利用公共前缀匹配来减少查询时间，尽可能减少多余的比较。



全局混合索引设计图

具体来说

**2.1.1 混合索引结构设计思想**

2.1．系统结构

系统架构如图，我们的分布式索引系统由四个模块组成。第一模块将文件分区成一定数量的block, 然后将他们分发到各个计算节点。文件包括很多文本数据，这些数据在分布式文件系统HDFS中收集。第二个模块中，每个计算节点分别从前一个模块获取分区块并构建本地的索引结构。所有索引树结构作为中间结果存储在HDFS上，并会缓存以便以后进行查询。

2.1.1分区策略的构建

由于文件中包含了海量的数据，需要尽可能比较均匀地分散到各个机器的节点上，因此分区的算法至关重要。Spark中的分区器决定了RDD中分区的个数，以及RDD中每条数据经过shuffle后，属于哪个分区，以及reduce的数量。

Spark的分区器默认是使用HashPatition，虽然实现起来比较简单，速度也比较快，但缺点是容易造成数据倾斜。因为Hash的本质是除留取余法，类似于 key.hashCode() % numPartitions。这样就容易造成同一个分区内数据量过多，而导致消耗大量线程资源，而其他的线程占用的较少。

在本课题中，根据实际需要，实现了RangePartitioner进行分区。RangePartitioner是一个范围分区器，它需要具备以下两个功能：首先它根据父RDD的数据特征，确定子RDD的边界。其次，给定一个key，能快速定位到其所在的分区。

RangePartitioner，是在TeraSort算法的基础上，采用水塘抽样的思想进行实现。它最适用于本课题之处在于，它解决了在数据总量很大且未知的情况下，且未将全部数据读入内存，甚至并不知道RDD中元素的个数时，能够做到等概率抽取数据。算法总体思路是，对父RDD进行采样，并将得到的数据排序，分成M个数据块。

1. 一般采样算法。

在不考虑父RDD数据情况下，暴力划分范围需要两次遍历RDD数据，这显然是非常麻烦的。在本课题，未知数据规模的情况下，我们可以借鉴水塘采样的方法。其核心要求是：要求最终返回元素有k个情况下，当取到第n个元素时，前n个元素被留下的几率相等，即k/n。

解决思想是：从待分类集合S中抽取首k项放入抽取集合C中，并生成从0到n范围内的一个随机数p，如果p<k，则把C中的第p项换成S[n]项。

伪代码是：

从S中抽取首k项放入C中

对于每一个S[n]项（n ≥ k）：

随机产生一个范围0到n的整数p

若 p < k 则把C中的第p项换成S[n]项

证明如下：

1.初始情况 n <= k：此时每个元素留下的概率均为1。

2.当 n = k+1 时，第k+1个元素留下的概率为k/(k+1)，前k个元素留下的概率均为：k/k \* (1 - k/(k+1) \* 1/k) = k/(k+1)，即上一步留下的概率乘以这一步留下的概率。

3.假设 n = m 时，每个元素留下的概率均为 k/n = k/m。

4.那么，当 n = m+1 时，第m+1个元素留下的概率为1/(m+1)，前m个元素留下的概率均为：k/m \* (1 - k/(m+1) \* 1/k) = k/(m+1)，其中：k/m为上一步留下来的概率，k/(m+1) \* 1/k 为这一步不能留下来的概率（第m+1个留下来，同时池中一个元素被踢出的概率）。

综上可知，算法成立。

1. 基于权重的再采样算法。

由于上游的父RDD数据量仍可能分布不均匀，在一些极端的情况下，一次采样后仅保证采样概率相同，却仍会存在有的分区占用数据量过大的情况，对于一些数据量不足的分区仍需要进行重新采样。

我们很容易得到这两个公式：

每个分区所需采样数=采样因子\*分区记录数。

采样因子 = 总采样样本数/总记录数。

情况1：如果采样因子\*分区记录数>分区内采样数，说明该分区采样数量不足，需要进行再采样。否则将其权重扩大为1/采样因子。

情况2：如果采样因子\*分区记录数<分区内采样数，说明采样数量过多，相应占有的权重会比平均权重大，这时权重为更新后的采样因子值。

根据样本权重重分区算法：

首先将采样出来的candidate(Array[(key,weight)])按照key排序，计算总权重sumWeights，除以分区数，得到平均权重step。

接下来while循环遍历已排序的candidate，累加权重sumWeight，达到step时就将获取到的key作为区间分隔符，最后返回所有的分隔符，这样就得到了边界。

（3）分区定位法

对于快速定位算法，若边界的数量少于128，则直接遍历分隔符，否则使用二分查找法找到合适的分区编号。

2.1.2索引树的构建

在分区策略成功构建后，数据已可以较均匀地分布到各个分区上，这时就需要构建索引结构了。我们输入的是带有特征向量的RDD，在每个分区节点上，执行mapPartition()将值迭代地插入索引结构中。人们普遍使用B树作为索引树，因为B树树高较短，能够提搞I/O速度。然而在这里由于spark基于内存计算，可以在主存中使用改进的CST树作为每个节点的索引结构，当然也可以换成其他树或者结构，选择多种方案进行比较。

CST树结构：

typedef struct TTREENODE         //树节点的结构

{

    TTREENODE \*left;             //节点的左子树指针

    TTREENODE \*right;            //节点的右子树指针

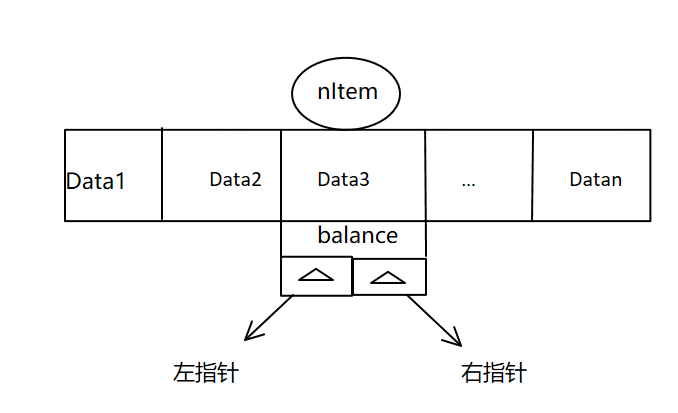
    unsigned short int nItems;       //节点中键值的数目

    ElementKey key[nItems];       //key值数组

ElementData value[nItems];     //对应数据行的指针数组

int balance;          //balance(平衡因子)，abs(balance)<=1，balance =右子树高度-左子树高度；

} TTREENODE;



Trie树相比hash方法，最大的特点是建树的过程即是查询的过程。Hash在少量数据时能达到O(1)的时间复杂度，但大规模数据面前很容易出现hash冲突，可以用链地址法处理冲突，得到的平均复杂度依旧是O(k)，k是冲突链地址的长度。

对于空间复杂度的分析：

首先设是数值属性个数，是其属性值的个数，是字符属性的个数，是其属性值的个数，在理想情况下，树是均匀分布的。设第一层索引的阶为k，字符的阶为m，N是总属性值个数，N =

在第一层，B+树，其树高是，其节点数为：

= 1+k++…+ = = (1)

在第二层，

对于CST树，由于其节点中存放key数组，平均的个数为nItem，所以在数据分布均匀时，需要的节点数为：

=1+m++…+= = (2)

对于Trie树，占用的节点数=

所以总的空间占用为:

+ \*

= + \* (3)

而对于传统的B+树索引，是为所有属性值建立索引，所以其占用的中存储空间为：

= + (4)

可以推导出来，(4)明显大于(3)。而且随着数据规模的扩大，传统索引所占用的空间越来越大。

对于时间复杂度的分析：

传统的B+树索引，总的查找的时间复杂度约为:

+

而混合索引，数值部分的时间复杂度约是：

字符部分的时间复杂度是:

T树查找算法：

查找即根据key值找data。从根节点开始，将key与这个节点最小值key[0]进行比较，如果key较小，则递归到左子树进行查询；否则将key与节点最大值key[nItems-1]进行比较，如果key更大，则向右子树遍历。否则说明key落在当前节点范围内，用二分法查找key相等的节点值，然后返回data数组中对应位置数据即可，否则查找失败。

查找算法代码如下：

TREENODE pNode = root;

While(pNode != NULL) {

If(nowKey <key[0]) {

pNode = pNode->left;

} else if (nowKey > key[nItems – 1]) {

pNode = pNode->right;

} else {

index = 当前key数组中二分法找到与key值相等的位置。

return pNode->data[index];

}  
}

由于父RDD的类型一般是String，在spark中定义一个新的数据结构，feature vector，用来保存每个特征向量的信息。这其中定义了id用来存储ID信息，定义features数组来存储数据维度。新生成的feature vector将替换之前的字符串。在每个计算节点上，建立索引树结构，并将RDD每个块的值插入索引结构中。

在索引树建成后，输出的T树集合将被存储到每个节点的内存中，用于检索。同时也要保证，输出与输入的分片数量相等。对于RDD中的每一维度的特征向量，在逻辑上属于同一分区，在集群上分布在不同的节点上。数据插入CST树的操作，是在各个节点上的本地操作，而在集群中同时并行执行。对于操作的中间结果，可以暂存到分布式文件系统HDFS上，防止一些突然断电带来的损坏。而且作为一种通用的格式存储，用户可以将中间结果直接加载到内存中，而不用重新构建树结构。

2.1.3分布式索引系统结构

分布式索引结构利用索引是用来支持基于Spark的对大规模数据的搜索技术。



2.1.3 谓词下推优化

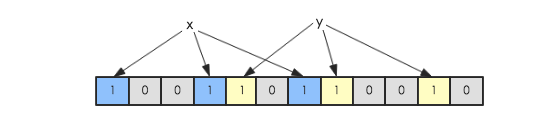
对输入的查询条件可以用谓词优化器来优化。谓词下推的基本思想是：尽可能早地进行数据过滤，以便更好地利用索引。在每一次查询树中，都尽可能早地先过滤掉部分数据，减少join等操作，以提高查询的速度。

Spark会对输入的sql语句进行语法分析（包括分词），包括词法分析，语法分析以及语义分析，并生成执行计划，主要谓词的优化在逻辑优化阶段生成，而物理优化则是RDD的DAG生成过程。

谓词下推的优化，是在sql执行的阶段，用谓词filter操作符进行过滤。如果无法有效地过滤数据，就会触发全局扫描，这样会造成大量重复数据。在这里可以使用bloomfilter过滤器，利用位数组对数据进行过滤，以进行优化。

布隆过滤器的原理是这样：

它使用多个独立的hash函数，将元素映射成多个哈希值，并将该bit位置1。



这样判断一个元素是否存在，只需要用多个hash函数映射，看对应位是否有0存在，若有，则说明这个元素不在集合中。

这样做好处有很多。第一，因为大量数据通过列式存储，列之间是独立存储的，使用布隆过滤器之后，如果某一列被过滤掉了，其对应同行的列就不需要扫描了，这样能减少扫描的次数。第二，可以减少联机之间网络的开销。这是因为大量重复数据被过滤掉，而不用通过socket发送到计算节点。最后，减少了hash join执行的开销。

2.1.4 结果推荐算法

结果推荐的目的是将查找最匹配的结果最优先呈现。由于数据集有许多不同维度的特征，在这里使用KNN分类算法。KNN算法有助于在T树中针对查询条件检索到k个最相似的结果。

首先，使用spark的flatMapToPair()方法，将每棵索引树返回key-value对象，然后再合并成一个Iterable。在查询前需要设缓冲区大小k，这个大小表明了，每个索引结构收集前k个相似的案例返回的<key,value>储存在新的RDD中。为了减少计算节点之间的shuffle运算，并提高并行计算效率，可以使用knn算法，来减少距离较远点所占的权重。

基于spark的并行knn相关度算法：

在HDFS中读取数据并建立RDD对象，然后对训练集分片。根据key读进每个map。利用余弦相似度来计算样本间距离。通过相似度计算，每个map都能得到最近的K个邻居，并将这k个邻居和其相似度保存成<class,distance>。每次做完map处理之后，会将中间结果转到reduce任务中。

算法过程：

输入训练集D，测试集T

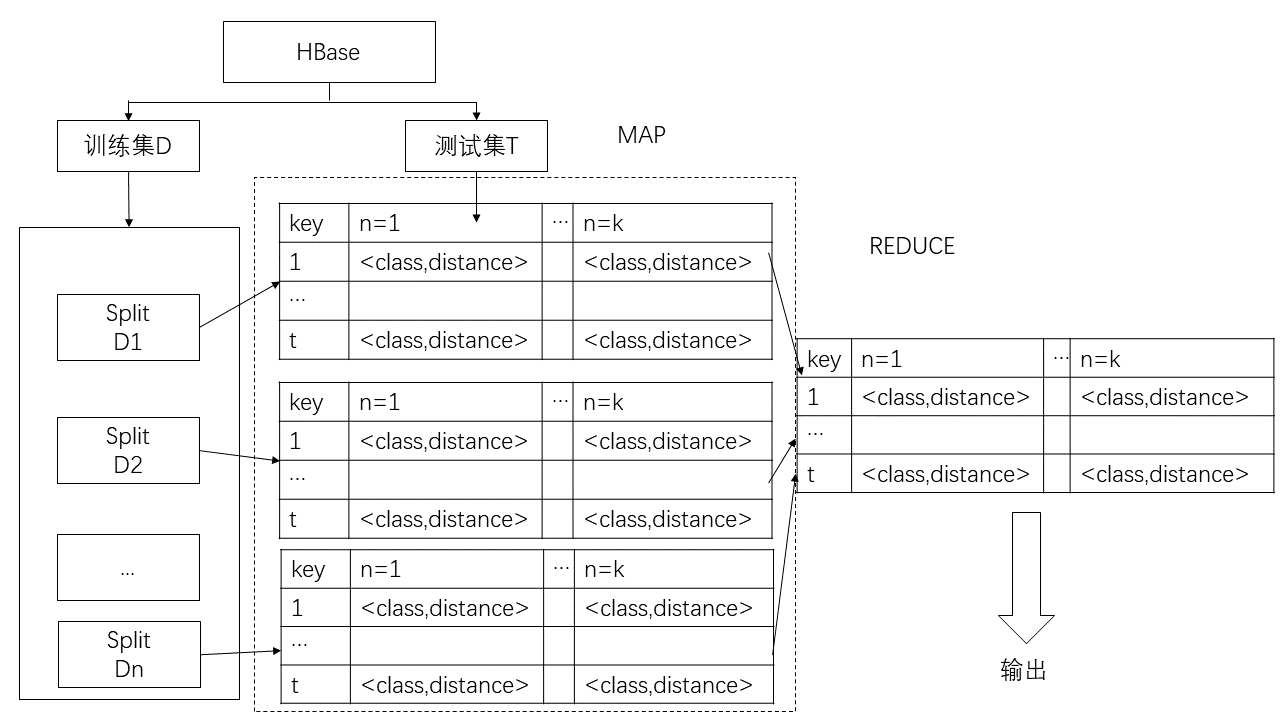
输出：分类情况。

计算过程：

1. 将训练集从Hbase读取并作为放入Map中。
2. 将测试集从Hbase读取并加入中
3. 对和进行数据预处理。
4. 在MAP过程：根据相似度计算方法，得到每个Map中最近的K个近邻，组成距离向量，装入value中。然后将结果发至Reduce
5. Reduce过程：

在Map中读取结果，对于每个元素，从最近邻居开始比较其与每个邻居的距离，并进入下一轮的迭代。如果map传来的距离小于当前的值，则用该值替换为新的距离值，否则进行下一轮比较。

使用基于TF-IDF的相似度计算方法，计算训练集与测试集样本间的距离，并得到K近邻。然后将这K个近邻组成<class, distance>装进value中。



Spark框架下RNN算法并行化过程

搜索的结果根据T加权相关度的分值高低进行排序，分值越高说明匹配的效果越好。基于词语权重的相关度计算方法如下：

首先是对距离计算的定义。这里距离是以KNN算法的TF-IDF计算为基础。因为字词的重要性随它在文件中出现次数正比增加，但却随着它在总索引库中出现频率成反比下降。

TF词频描述的是词条在文件中出现的次数，

其中是词在文件中出现的次数，分母是中所有词频出现次数之和。

IDF是逆文档频率，是用总文件数/包含该文件的文件数量，再把商取对数。

这里考虑到数据本身的相关性，以及分布式集群的情况，引入变量p作为分区系数，并重新定义了权重为：

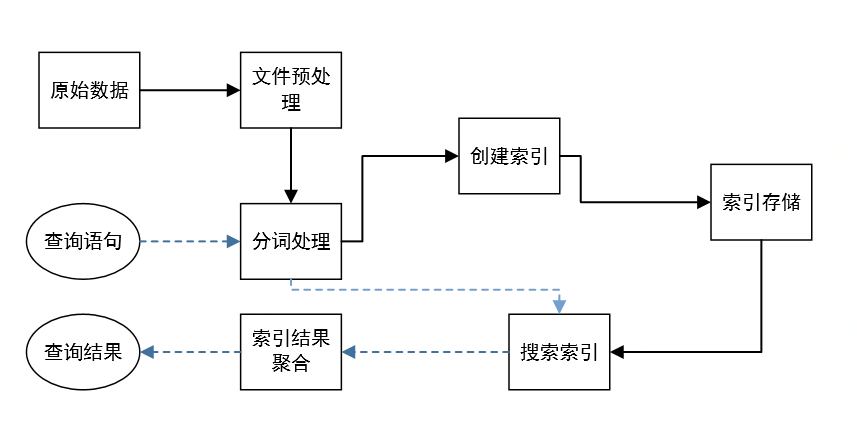
其中，k表示文档中除词条之外的其他词条个数。

2.2.1 检索系统流程

分布式混合索引检索系统，其检索流程如图所示。首先是根据原始的数据建立词条的索引来进行大规模数据的检索。基于分布式检索的架构，将检索系统分为三个子部分：数据预处理系统、索引创建系统、搜索算法系统。

数据预处理系统，主要是将原始的数据进行数据清洗、数据提取、编码转换等数据操作，生成统一编码格式的文件，并经过数据清洗，作为创建索引系统的输入。

创建索引系统，是将预处理后的数据，经过MapReduce进行分词，过滤，



2.2.1 分布式协调系统

执行索引创建和查询的Spark集群基使用Zookeeper作为分布式协调系统，维护集群的分布式同步和负载均衡。Zookeeper集群之间通过zab协议保持数据一致性，该协议分为两个阶段：leader选举和原子广播。选举算法采用少数服从多数原则，当leader宕掉之后，会重新选举出leader。原子广播会同步leader和follower之间的消息，保证集群之间状态的一致性。

索引集群的选举算法：

k:机器数量

function leader\_select()

For m <— 1 to k do

Message = Vote\_for\_myself(m)

Broadcast(message)

If Getreply() do

Vote\_number = Compare ()

If(vote\_number > sum\_machine\_num/2)

Status = leader

Return m

End if

End if

End for

机器1启动，给自己投票，并广播和侦听。由于没收到反馈，一直处于Looking状态。

然后机器2启动，给自己投票，广播时与机器1交换结果，由于2号编号大所以2胜出，但由于投票数没过半，所以状态仍是Looking。

之后机器3启动，给自己投票，并广播与机器1，2交换信息，由于3大3胜出，此时投票数过半，所以3成为leader。

负载均衡的实现：

集群中的机器，一旦有节点宕掉，watcher会感知到机器状态的变化，作为动态注册和进行服务配置中心，可以拿到对应的服务器状态，并进行集群的重新选举。由于zookeeper的zab协议，可以保证信息的一致性。其选举机制也可以让我们比较方便地扩容。

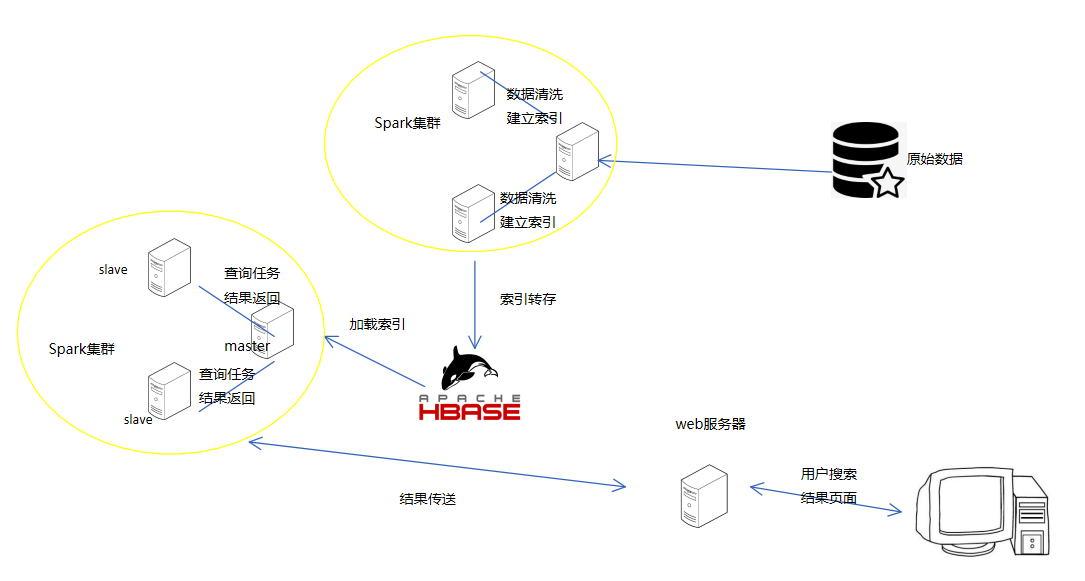
当一台新的机器启动时，会将提供的服务名称、服务地址注册到服务配置中心，服务消费者可以通过中心获得NameNode下机器的列表节点。然后通过选举机制重新选举leader，并通过原子广播，同步数据信息。

2.2 分布式混合索引平台框架

在分布式检索平台中，首先利用spark的Map、filter、split等算子，对原始数据进行一些预处理和清洗，提取文本内容，进行分词和处理。然后根据其结构，建立混合结构的索引。之后将索引表由HDFS转存到HBase中。HBase是基于列存储的数据库，其数据压缩性比HDFS强一些。而且由于查询比较频繁，用HDFS会经常出现全表扫描，而HBase可以实现随机读取和小范围扫描，加快读取速度。存至HBase的索引会根据其Region分布到集群不同节点上，当数据增长时，会自动分割region并重新分布。

系统建立索引的过程如下 ：

1. 集群先利用MapReduce框架对原始数据进行处理，生成索引的原始文件。每个索引目录下有许多shard索引文件。每次生成索引时，集群会向master节点汇报生成的消息通知。
2. HBase的Hmaster节点从HDFS存储系统拷贝索引文件，再根据负载平衡机制将索引文件分配到各从节点的机器上，建立备份。



使用混合索引系统搜索的过程如下：

1. 客户端通过搜索界面，在浏览器提交搜索请求。
2. Web服务器通过调用搜索接口函数，向spark集群发起搜索请求。
3. Spark的master节点接收到索引请求，将查询需求广播到spark集群的从slave点上。
4. 各slave节点在本地执行搜索任务，并将符合条件的搜索结果返回给master。
5. Master对搜索结果做reduce处理，并将结果返回给web服务器。
6. Web服务器将查询到的结果返回给客户端。

2.2 索引管理

2.2.1 接口函数

这里将spark的一些函数进行改写，来实现索引的功能。

* 创建索引。根据索引的位置找到索引文件，按索引结构建立索引，并用索引名进行命名：

Void createIndex(String indexName,String indexPath);

* 删除索引。根据索引名，找到集群中索引的位置，删除索引，并调整索引树结构：

Void removeIndex(String indexName)

* 列出集群中所有索引

Void listAllIndex()

* 集群中分布式检索索引

Void search(String queryWord)

* 合并索引。读取旧索引文件的路径，合并生成新索引文件

Void mergeIndex(String newName,String[] indexNames)

2.3 任务管理

由于Hadoop集群调用Spark索引模块处理原始数据，每次生成一个索引文件时，都会向集群的master节点发送消息，定义格式为“索引路径，索引名，主节点IP：端口号”。Spark主节点接收到信号后，将HDFS里的索引文件复制到HBase，并将其部署到Spark集群的各从节点中。

索引检索系统处理海量规模的数据，而且每时每刻都有大量索引文件的产生，需要将一些小文件合并，以此提高检索效率便于管理。索引平台会使用定时器，每隔一小时调用mergeIndex函数将属于同一段的索引文件合并，合并后删除原索引。

实际操作时，由于索引合并需要十多秒的时间，如果这时候客户端执行搜索任务，可能发生冲突，从而造成检索结果的遗漏。这时可以在HDFS里建立两个变量，“/spark/UPDATE”为旧的索引目录，“/spark/SEARCH”为新的索引目录。初始时变量都为0，在合并时将“/spark/UPDATE”置为新文件名，“/spark/SEARCH”置为旧文件名。在合并完成后，将变量重新赋值为0。这样可以避免冲突。

2.4 评价指标

评价可以使用一些常见的指标。对于搜索的评价，可以用准确率与查全率评价。准确率可以表示分类的正确性，即正确搜索到的文档数量占全部检索到文档的总数。查全率就意味着分类的完整性，也就是正确检索到的文档数量占所有应被正确检索到的文档数量的比例。公式如下：

Spark的分布式并行计算的加速比可以评价为：

加速比 = 串行一个节点计算时间/n个节点并行计算所用时间

使用加速比可以衡量系统的扩展性能。

3 实验

3.1 实验环境搭建

通过VMware创建了3台虚拟机作为集群环境。然后安装Hadoop组件，作为基础。然后搭建Zookeeper集群，作为分布式应用的协调服务程序。再以此为基础安装Spark框架，集群部署采用Standalone模式，并且使用Hbase作为分布式数据库。

从便于配置的角度出发，将服务器主节点配成Spark和HBase的主管理节点，并将从节点配成Slave1和Slave2。

主要参数如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 硬件环境 | | 软件环境 | |
| 硬盘 | 150GB | 操作系统 | Centos 6.5 |
| 内存 | 8GB | Java版本 | Java-1.8.0 |
| CPU | rhel6 | Spark版本 | 2.2.1 |
| 网卡 | 桥接虚拟网络 | Hadoop版本 | 2.0 |
|  |  | Zookeeper版本 | 3.4.5 |
|  |  | HBase版本 | 2.0 |

分布式索引平台的部署方法如下：

1. 配置各机器的Java环境，创建ssh免密连接，并修改机器hosts。
2. 配置Hadoop环境，修改环境变量和配置文件，在bin下运行start-all.sh启动文件。
3. 部署Spark集群，按下面方法执行：

首先将在各节点安装Spark环境，然后修改Spark配置文件。将管理主节点配置为master，从节点配置为slave。

spark.master spark://192.168.0.1:7077

spark.executor.memory 512m

spark.eventLog.enabled true

spark.serializer org.apache.spark.serializer.KryoSerializer

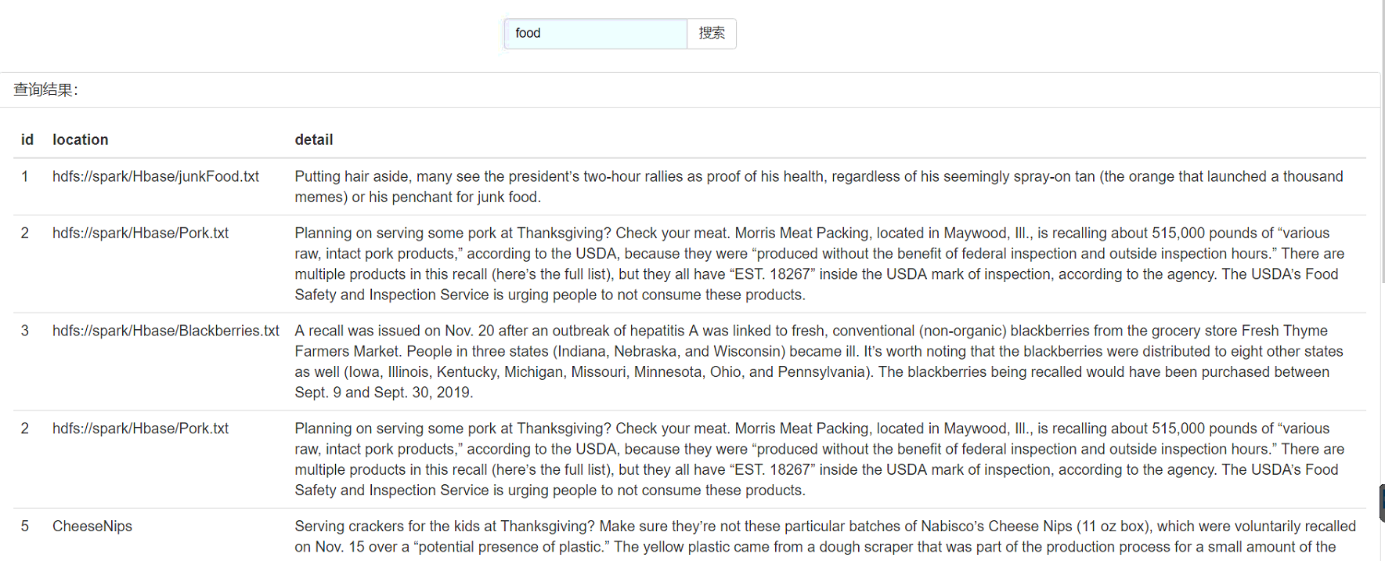
在sbin下分别启动./start-master.sh和./start-slave.sh

1. 然后安装和配置zookeeper。将zookeeper.servers配为master:2181，与spark的主节点相同。启动zk服务：sh bin/zkServer.sh start
2. 安装HBasse，将hbase.rootdir根目录 设为hdfs://spark/Hbase，修改regionservers为slave1和slave2等。然后切换到bin目录下使用./start-hbase.sh启动hbase。
3. 开发索引系统。在集群的基础上，根据索引策略开发索引结构。
4. 在宿主机上安装Tomcat作为web服务器，使用Spring boot框架开发web应用，调用索引系统的接口，进行查询和结果的展示。

3.2 对于交互web页面设计

交互页面主要基于Java web。用tomcat搭建web服务器，使用Spring框架与用户交互，调用索引集群的接口进行索引查询。

用户在搜索框内提交关键词后，可以搜索到相关的文本内容，以及文本所在的路径目录。



2.5实验分析

经过实验测试发现，系统可以正常运行，能够对数据进行接收，并建立索引系统，与用户界面交互后，可以去集群中检索，并将请求到的结果反馈回客户端。

通过在客户端编写批量测试程序，求取平均值，得到的索引集群性能指标如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据量规模 | 4MB | 118MB | 973MB |
| 索引建立耗时 | 2.278s | 29.383s | 223.329s |
| 索引建立速度 | 1.756MB/s | 4.016MB/s | 4.357MB/s |
| 千次搜索时间 | 43.432s | 64.321s | 453.325s |
| 索引文件大小 | 2.3MB | 32.3MB | 353.2MB |
|  |  |  |  |
|  |  |  |  |

由表可以看出，索引的建立和部署速度都随着数据量的增加而加快，这是因为集群的批量处理优势被放大，集群的资源调度和任务分发的时间占比减少了。相应的检索效率更高。

3．后期拟完成的研究工作及进度安排

2019.12~2020.3：完成其他几种索引方法的实现。

2020.4~2020.5：优化索引结构，提高系统效率。

2020.5~2020.6：总结课题研究结果，撰写结题报告，准备毕业答辩。

4．存在的困难与问题

基于Spark的分布式索引系统中涉及一些较难的算法知识，需要仔细推敲；同时对工程实践能力也有一定的要求。此外，系统整体的性能也需要进行进一步的优化，许多系统指标需要进行调试和优化。

可能遇到数据倾斜的问题。即由于某些标签的数据较多，分配到一个Datanode，会造成单一节点的负载过重，而其他节点却相对稀疏。解决方法是对每个Datanode设立一个阈值，如果超出这个阈值，就把数据标记，并分配到别的Datanode上，同时在索引表中将该数据标签与原数据标签进行合并。

5．如期完成全部论文工作的可能性

论文已经完成了超过一半的工作，虽然剩下的时间也比较紧张，但我相信只要严格按照计划进行，可以完成全部论文工作。