浙江大学

课程名称 暑期实践报告

题　　目 基于 Elasticsearch 的垂直搜索引擎

姓　　名 陈立华

学　　号 3150102577

摘要： Elastic Search(以下简称 ES)是一个基于 Lucene 的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎。Elasticsearch 作为 Apache 许可条款下的开放源码发布，开发语言为 Java。是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。基于上述特性，针对本次实验中，一个垂直所有引擎，我们采用 Elastic Search 作为存储数据的数据库（本质上，ES 可作为一个数据库），将爬虫爬取的网页内容存储在 ES 中，当用户发起关键字搜索请求时，通过 ES 强大的全文检索能力，将相关的结果返回

关键字 Elastic Search　垂直搜索引擎 全文检索 旅游

一 前言

当我们面临建立一个网站或应用程序的需求，添加搜索功能通常是必选项，我们从大二下学期开始，一直到大三上学期，所用的教学服务系统，在主页面上一般都会放置一个搜索框，但之前一般都是一个伪搜索框，但在此时，我们将通过 ES 来实现它。但是想要完成搜索工作的创建是非常困难的。我们希望搜索解决方案要运行速度快，我们希望能有一个零配置和一个完全免费的搜索模式，在搜索语句上，我们希望能够简单地使用 JSON 通过 HTTP 来索引数据，我们希望我们的搜索服务器始终可用，我们希望能够从一台开始并扩展到数百台，我们要实时搜索，我们要简单的多租户，我们希望建立一个云的解决方案。因此我们利用 Elasticsearch 来解决所有这些问题及可能出现的更多其它问题。题主曾在公司实习时，针对公司不同业务层面的业务会用不同的数据库来进行存储，题主当时在大数据部门，因此，对于预聚合显得尤为迫切，以 Druid 这种 OLAP 形式的数据库作为主数据库进行存储。同时，针对一些细分的需求，比如调用链，对于插入速度显得有迫切需求的，则采用了 Cassandra，而对于公司的核心业务层面的数据，则希望能做到原文记录，并在有需求的时候，进行原文检索，因此公司选择了 ES 作为了原文存储数据库。

二 ES 简介

1 基本概念

Elastic 本质上是一个分布式数据库，允许多台服务器协同工作，每台服务器可以运行多个 Elastic 实例

Node：单个 Elastic 实例

cluster：一组节点构成一个集群

Index：Elastic 会索引所有字段，经过处理后写入一个反向索引（Inverted Index）。查找数据的时候，直接查找该索引，数据管理的顶层单位。可简单地理解为数据库。

Document：Index 里面的单条记录成为文档，许多条文档构成一个 Index。文档使用 JSon 格式表示

Type：将 Document 根据不同的性质进行分组。这种分组就叫 Type。

可通过一张表来简单与 MySQL 进行类比

Elasticsearch	MySQL
index（索引，名词）	database
doc type（文档类型）	table
document（文档）	row
field（字段）	column
mapping（映射）	schema
query DSL（查询语言）	SQL

Elasticsearch 就是一款面向文档的 NoSQL 数据库，使用 JSON 作为文档序列化格式。但是，它的高级之处在于，使用 Lucene 作为核心来实现所有索引和搜索的功能，使得每个文档的内容都可以被索引、搜索、排序、过滤。同时，提供了丰富的聚合功能，可以对数据进行多维度分析。对外统一使用 REST API 接口进行沟通，即 Client 与 Server 之间使用 HTTP 协议通信。

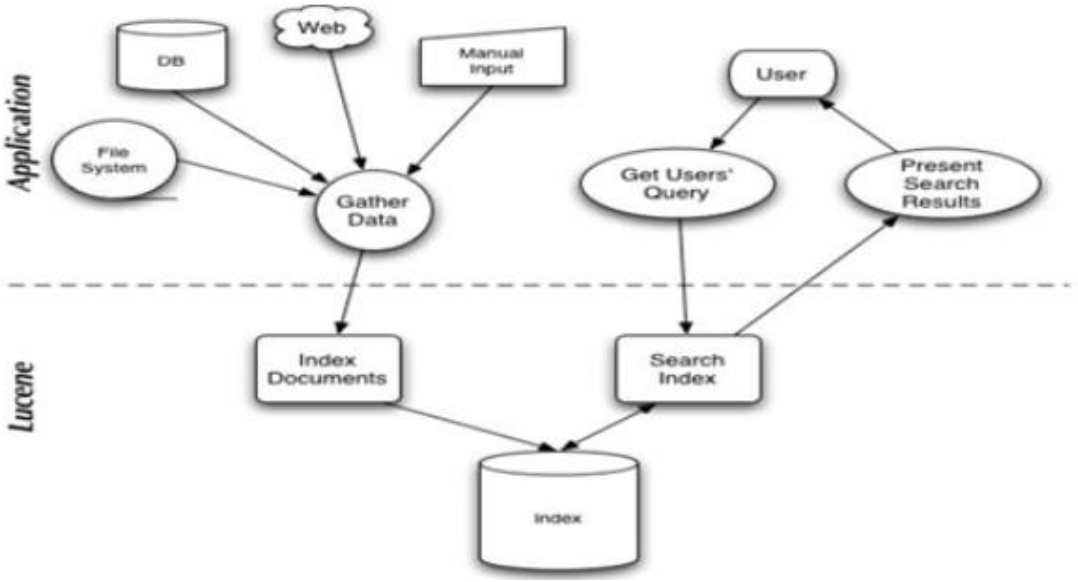
2 ES 实现全文检索原理

因为 ES 以 Lucene 作为核心，因此，在此处将 Lucene 具备的若干性质在全文检索中的作用进行描述。

Lucene 主要包括两块:

一是从文本内容切分词后索引入库;

二是根据查询条件返回结果，即建立索引和进行查询两部分。

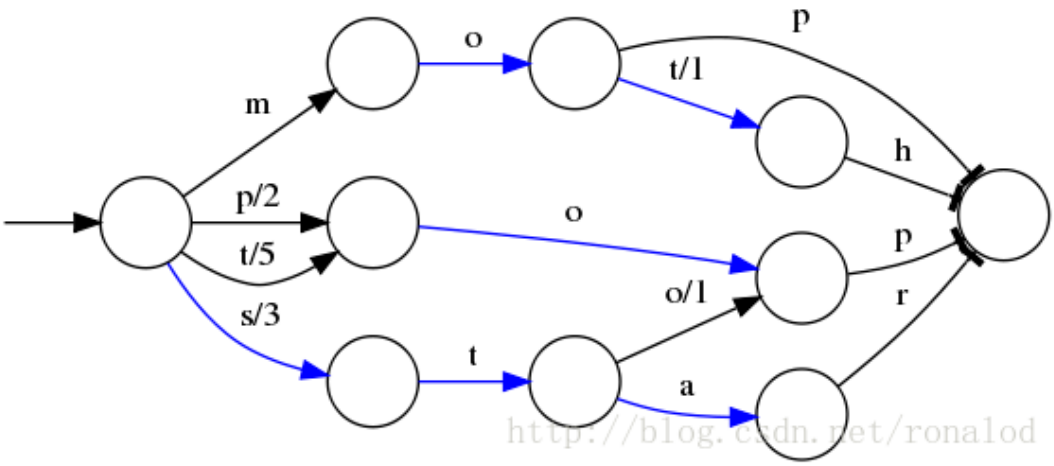


2.1 索引原理

Lucene 全文检索基于倒排索引来实现。其中倒排索引可以分为两部分，词典和倒排表。

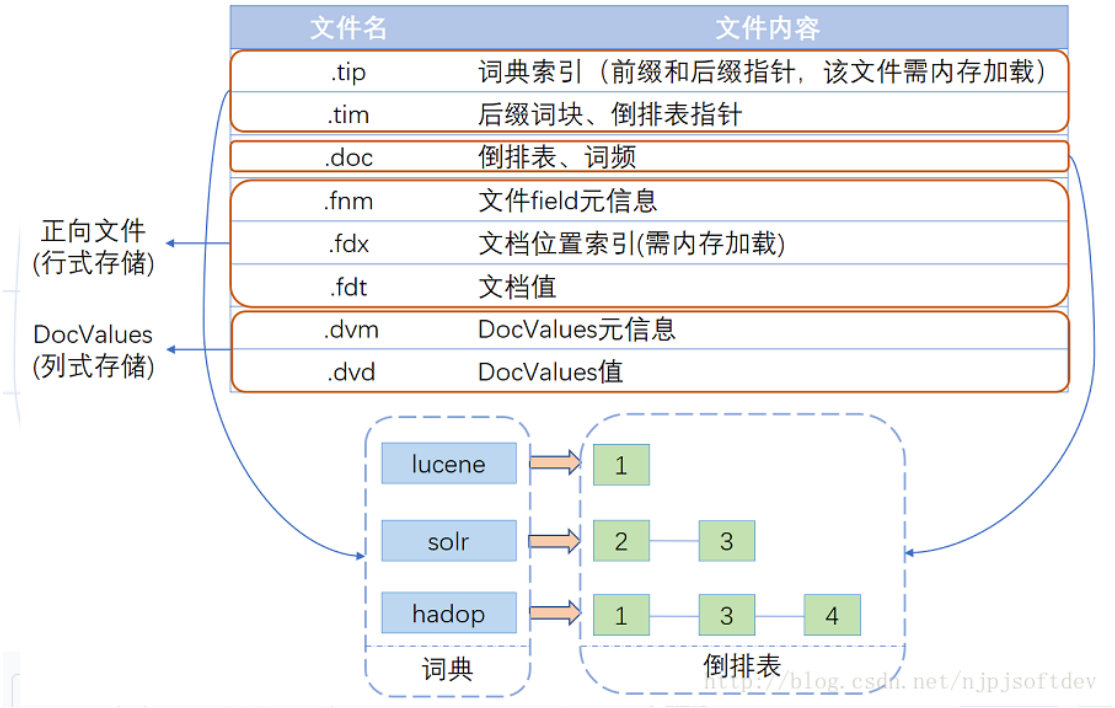
其中词典结构可以有多种结构，包括 B+树，调表，FST（Finite State Transducers）

Lucene 现在所用的结构为 FST



通过共享前缀的方式，内存占用率低，压缩率一般在 3~20 倍之间，模糊查询支持好，查询快。但结构复杂，输入要去有序，更新不易。

2.2 索引实现



检索过程分为三个步骤

1. 内存加载 tip 文件，通过 FST 匹配前缀找到后缀词块位置。
2. 根据词块位置，读取磁盘中 tim 文件中后缀块并找到后缀和相应的倒排表位置信息。
3. 根据倒排表位置去 doc 文件中加载倒排表。

2.3 倒排表结构

倒排表是文档号集合。Lucene 现使用的倒排表结构叫 Frame of reference,它主要有两个特

点：

1. 数据压缩

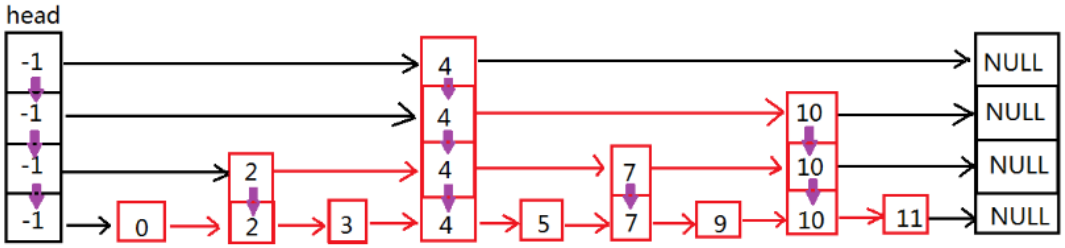
2. 跳表加速合并，因为布尔查询时，and 和 or 操作都需要合并倒排表，这时就需要快速定位相同文档号，所以利用跳表来进行相同文档号查找。

跳表：跳表全称叫做跳跃表，简称跳表。跳表是一个随机化的数据结构，可以被看做二叉树的一个变种，它在性能上和红黑树，AVL 树不相上下，但是跳表的原理非常简单，目前在 Redis 和 LevelDB 中都有用到。

在对有序顺序表进行搜索时，使用二分查找时间复杂度是 O(logn)，但是有序顺序表的插入和删除却是 O(n)的算法。

在对有序链表进行搜索时，时间复杂度是 O(n)，但是对链表的插入算法却是 O(1)。

跳表模型：



2.4 正向文件

正向文件指的就是原始文档，Lucene 对原始文档也提供了存储功能，它存储特点就是分块+压缩，fdt 文件就是存放原始文档的文件，它占了索引库 90%的磁盘空间，fdx 文件为索引文件，通过文档号（自增数字）快速得到文档位置

fdt 为文档值，里面一个 chunk 就是一个块，Lucene 索引文档时，先缓存文档，缓存大于 16KB 时，就会把文档压缩存储。一个 chunk 包含了该 chunk 起始文档、多少个文档、压缩后的文档内容。

fdx 为文档号索引，倒排表存放的时文档号，通过 fdx 才能快速定位到文档位置即 chunk 位置，它的索引结构比较简单，就是跳表结构，首先它会把 1024 个 chunk 归为一个 block,每个 block 记载了起始文档值，block 就相当于一级跳表。

查找文档可分为三步：

第一步二分查找 block，定位属于哪个 block。

第二步就是根据从 block 里根据每个 chunk 的起始文档号，找到属于哪个 chunk 和 chunk 位置。

第三步就是去加载 fdt 的 chunk，找到文档。

2.5 结果排序：

计算 Term 权重

1.Term Frequency (tf):即此 Term 在此文档中出现了多少次。tf 越大说明越重要。

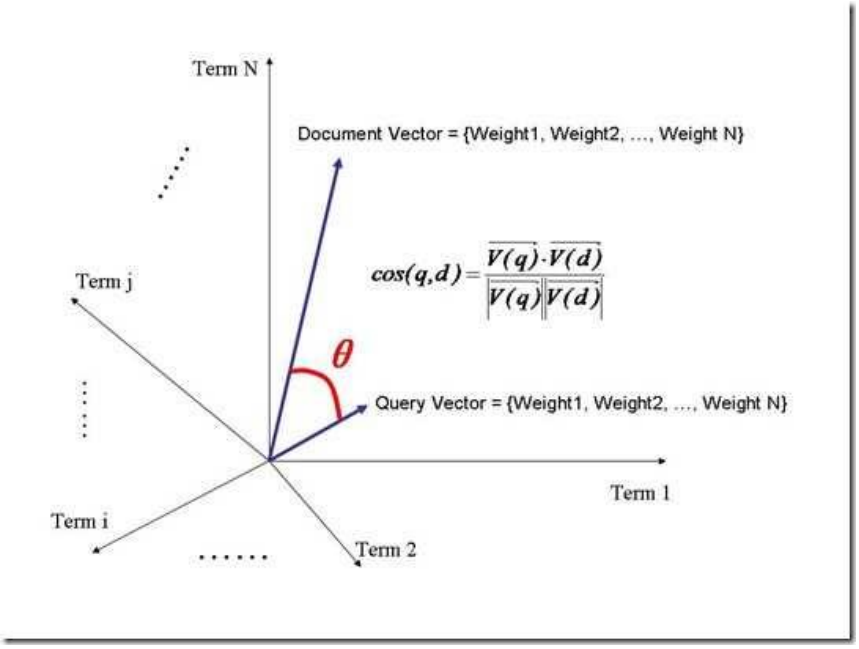
2.Document Frequency (df):即有多少文档包含次 Term。df 越大说明越不重要。

$$w_{t,d} = tf_{t,d} \times \log(n / df_t)$$

$$w_{t,d} = \text{the weight of the term } t \text{ in document } d$$
$$tf_{t,d} = \text{frequency of term } t \text{ in document } d$$
$$n = \text{total number of documents}$$
$$df_t = \text{the number of documents that contain term } t$$

Vector space model（VSM）
把所有此文档中词(term)的权重(term weight) 看作一个向量。
Document = {term1, term2, ... ,term N}
Document Vector = {weight1, weight2, ... ,weight N}
把查询语句看作一个简单的文档,也用向量来表示。
Query = {term1, term 2, ... , term N}
Query Vector = {weight1, weight2, ... , weight N}
取二者的并集,如果 Document 不含某个词(Term)时,
则权重(Term Weight)为 0

$$score(q,d) = \frac{\vec{V}_q \cdot \vec{V}_d}{\left|\vec{V}_q\right|\left|\vec{V}_d\right|} = \frac{\sum_{i=1}^{n} w_{i,q} w_{i,d}}{\sqrt{\sum_{i=1}^{n} w_{i,q}^2}\sqrt{\sum_{i=1}^{n} w_{i,d}^2}}$$



（综合利用了协同过滤的思想）

综合来看:
Lucene 的底层架构可概括为:

实现索引（FST）
倒排表（FOR 、Skiplist）
结果合并（链表合并算法）
正向文件（DocValues、LZ4）
结果打分（VSM）

三 项目中的 ES

3.1 项目确定

本次暑期的实践课程，课程内容要求为一个垂直搜索引擎，我们综合考虑了每一个细分行业的垂直搜索引擎，从一开始的租房行业，这个当我们打开链家网的时候，发现链家已经对这个接口有了非常完善的功能。然后，因为题主之前自己在一家旅游公司实习，公司自己内部实现了一套旅游业务的线上化，同时，对旅游线路提供了一个搜索功能。但当题主在使用它的搜索功能的时候，发现根据线路推出来的线路不是很符合搜索的本意，而且，在自己的系统之内，搜索结果可以有很多的改进地方，比如，很多相关性很显然，很高的线路没有出现在结果中。
因此，在最后的确定过程中，我们打算以旅游线路作为突破口，做一个旅游线路的垂直搜索引擎。

3.2 技术栈实现

我们打算将我们的搜索窗口用于支持一定数据量的线路的搜索。因此，我们在项目框架的最初实现了一个 JAVA 爬虫，用于爬取若干旅游网站的旅游线路数据，同时，对网站上的数据进行适当的解析，按照日期，价格，地点，标签（旅游线路的标签，比如亲子游，户外游等，这也是最终推荐模块的基础）等，将线路存到数据库中，在项目中，数据库为之前介绍的 ES。最后，通过前端，界面化地实现搜索框，当用户在搜索框内输入关键字后，ES 会将之前存于库中的若干条符合条件的数据呈现，返回给前端。同时，系统会基于用户的输入行为，（初期行为或者若干次行为）为用户推荐相似的线路，基于协同过滤为用户推荐口味相似的产品。并且，随着用户输入数据的增加，系统的相似度矩阵会越来越完善。推荐的结果也会越来越符合用户的需求。

四 参考资料及文献

https://www.elastic.co/products/elasticsearch（ES 官网）