基于 Map Reduce 的 Wikimedia dump 信息检索系统

学号：16307130194，姓名：陈中钰

1 项目概况

项目建立了一个完整的信息检索系统，包括数据处理、索引建立、信息检索部分，并搭建了对应的检索网站。

项目首先对 Wikimedia dump 数据做基本处理，接着使用 Map Reduce 建立以 ID 排序、以 TF,ID 排序的倒排索引，并在建立索引的同时进行索引的无损压缩，另外也尝试了实现有损压缩。此外，项目利用 Map 的过程建立了文章信息的索引。而为了降低检索时的内存压力，项目还利用 Map 的过程在索引上建立索引。最后，项目实现了多个词的排序检索，使用了 query-at-a-time 和 document-at-a-time 两种检索打分方式。另外，项目实现的检索网站可以对输入的内容进行检索，返回排序好的文章结果，还可以通过对应的链接访问文章内容。

索引建立部分使用 Java 语言和 hadoop 2.6.4 实现，而检索部分使用 Python 语言实现。另外，检索网站也是基于 Python 语言实现的，使用了 Flask 库。

2 项目设计

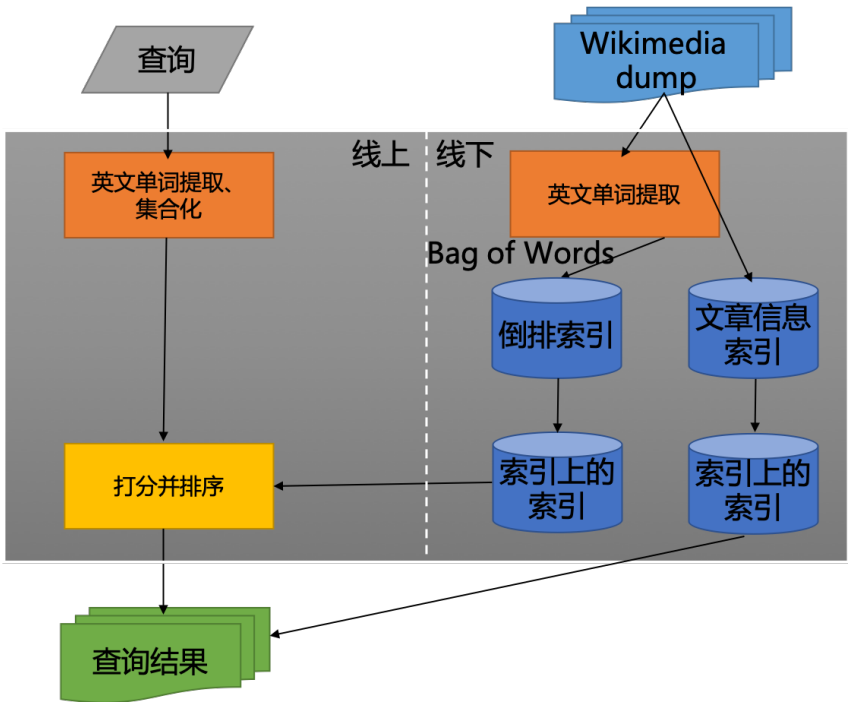

Figure 1: 信息检索系统设计

信息检索系统设计如 Figure 1所示，主要包括线下执行的索引建立过程、线上执行的检索过程。

在线下索引建立部分中，首先要对 Wikimedia dump 数据进行处理，取出里面的文章，并过滤出英文单词。获得的 bag of words 通过 Map Reduce 过程建立倒排索引，通过 Map 过程在倒排索引上再见一层索引，通过增加硬盘读取来降低内存要求。在数据上还要通过 Map 过程建立文章信息的索引，使得可以通过文章 ID 快速获得文章信息及内容。在文章信息索引上同样也可以通过 Map 过程建立索引。

在线上检索部分中，首先要处理查询串，获得其中的英文单词。再通过倒排索引，获得单词对应的文章 ID、单词出现次数，并使用不同的打分方法对相关文章打分，并按照分数对文章进行排序，最后输出检索结果。通过检索网站可以提供查询入口，并可以对检索结果进行可视化。

3 实现方法

3.1 倒排索引

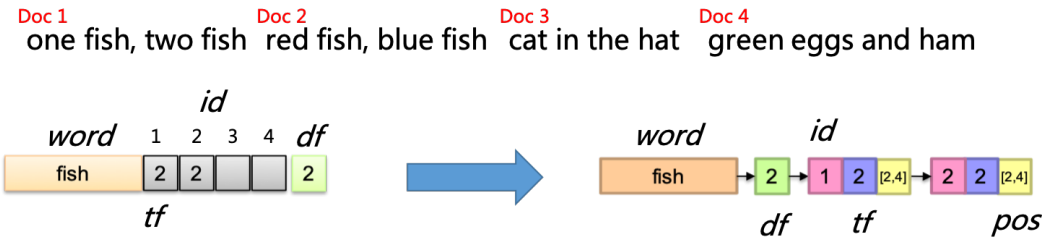

Figure 2: 倒排索引

如 Figure 2所示，倒排索引要实现的就是从单词到文章的映射。单词对应的多个文章以链表的形式存储，需要记录包括文章 ID、单词在文章中出现的次数 TF 以及对应的位置 position。在检索查询项的的时候，可以通过倒排索引快速获得查询项中的单词对应的文章，并根据记录的 TF、position 信息可以对文章打分并进行排序，进而获得分数高的文章，理想情况下就是查询项最相关的文章。

3.1.1 以 ID 排序的倒排索引及索引压缩

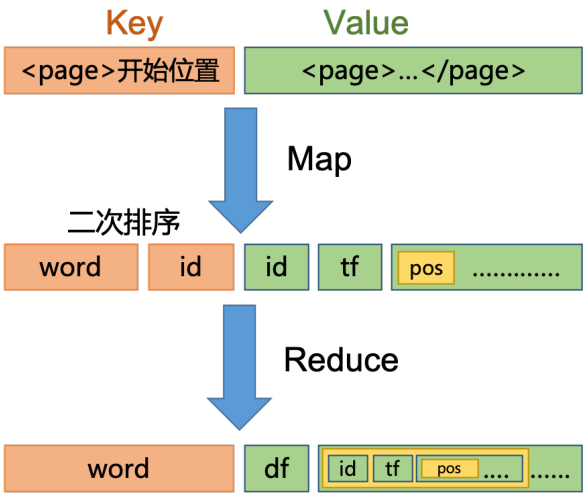

Figure 3: 以 ID 排序的倒排索引的 Map Reduce 过程

以 ID 排序的倒排索引的 Map Reduce 过程如 Figure 3所示。利用了 Map Reduce 会对 key 进行排序的性质，来实现二次排序，使 word 对应的文章链表的 ID 是从小到大排序的。

Map 阶段输入的 key 是文章 <page> 的开始位置，而 value 是文章在 <page>、</page> 之间的信息。在这个阶段，需要用正则表达式从 value 中取出文章 ID、文章内容 text，接着用正则表达式从文章 text 中匹配出单词，并获得单词在文章 text 中的偏移量 position。接着用 hash map 统计各个单词在文章 text 中出现次数 TF 和对应的位置 position。最后以单词、ID 为 key，以 ID、TF、位置列表为 value，发送给 Reduce 阶段。这里实现了二次排序，通过构建包括单词、ID 的 WritableComparable 类，定义以单词为第一排序键值、ID 为第二排序键值，那么在 Shuffle and Sort 阶段，hadoop 就会对单词、ID 进行排序，生成的文章流按照 ID 从小到大的顺序输入 Reduce 阶段。

Reduce 阶段输入的是相同单词、ID 排好序的文章信息流。按照顺序，把文章信息拼接起来，以单词为 key，以文章数量 DF、文章 ID、TF、位置信息为 value，写到索引文件中。这样就生成了以 ID 排序的倒排索引。

使用二次排序利用了 hadoop 的 Map Reduce 在 Shuffle and Sort 阶段会对 key 进行排序、分区的性质，使 hadoop 的 Map Reduce 框架对 ID 进行排序，从而避免了在 Reduce 阶段对文件信息流进行排序。文件信息流进入 Reduce 阶段的时候，是以流的形式进入，每次只会进入一个信息点，而不需要缓存整个信息流。所以，通过二次排序，不仅可以省去排序所需的大量时间，而且不需要把整个信息流缓存下来，能显著地降低内存要求。

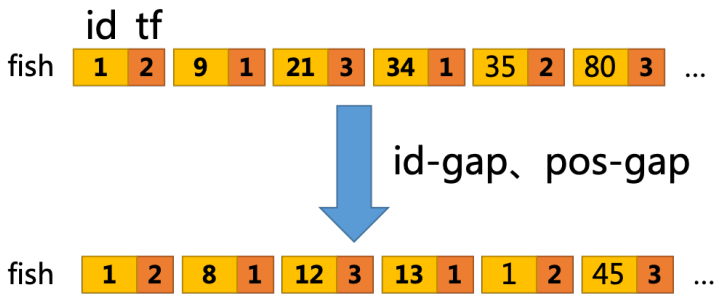

Figure 4: 有序的 ID、position 以 gap 的形式存储

此外，通过二次排序把单词对应的文章信息按照 ID 排序后，可以把 ID 以 gap 的形式存储，如 Figure 4所示。由于上述 key-value 索引是以字符串的形式存储的，那么以 gap 形式存储可以使 ID 长度变短，也就使索引大小显著减小。而在检索的时候，可以容易地把原文章 ID 恢复。

另外，每个文章中的 position 信息本来就是有序的，所以也可以把 position 以 gap 的形式存储，可以进一步地压缩索引。在检索的时候，同样可以简单地把原 position 信息恢复。

3.1.2 以 TF,ID 排序的倒排索引和索引压缩

以单词、ID 为 key 的二次排序可以简单地转化为以单词、TF 为 key 的二次排序，这样就可以生成以 TF 从大到小排序的倒排索引。另外，由于 power laws 的现象，有较小的 TF 的文章数量多，因此具有相同 TF 的文章数量也就多了。所以可以在 TF 从大到小排序的基础下，在 TF 相同时，可以对 ID 进行排序。和以单词、TF 为 key 的二次排序类似，通过把单词、TF、ID 作为 key，构建对应的 WritableComparable 类，并定义 ID 为蛋散排序键值，可以实现三次排序，如 Figure 5所示。这样在 Reduce 阶段所输入的文章信息流就是具有相同单词、按 TF 从大到小排序、TF 相同时按 ID 从小到大排序的，最后以单词为 key、其他信息为 value，就可以获得按照 TF、ID 排序的索引。

在 TF 相同的时候，ID 是从小到大的，可如3.1.1中一样，存储 ID 的 gap，如 Figure 6所示，可以很好地实现索引压缩，在检索的时候，也同样可以较容易地把原 ID 信息还原出来，

此外，如3.1.1中一样，position 信息本来就是有序的，也同样可以改为存储 gap，可以进一步压缩索引。在检索的时候，可以轻易还原出原 position 信息。

上文中所提到的压缩方式都是无损压缩，可以根据规则完全还原出原数据的。此外，还可以尝试进行有损压缩。根据主要英语词典，最长的单词只有 45 个字符，因此可以考虑忽略超过 50 个字符的单词。另外，一个单词的倒排索引已经按照 TF 进行排序了，TF 大的文章相对来说会更重要，而 TF 小的文章则没那么重要，所以可以考虑把单词的文章链表中 TF 小的后部分去掉。

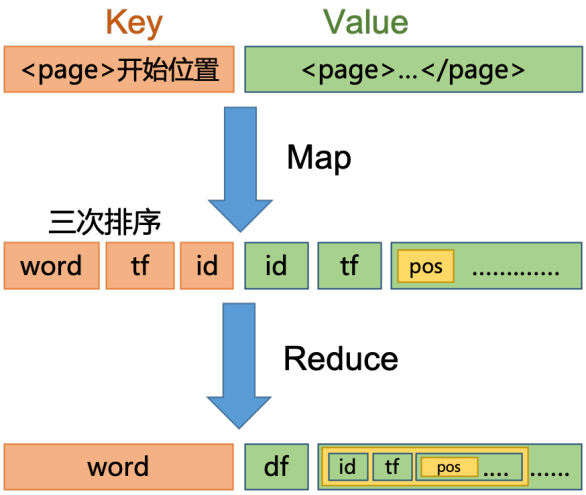

Figure 5: 以 TF,ID 排序的倒排索引的 Map Reduce 过程

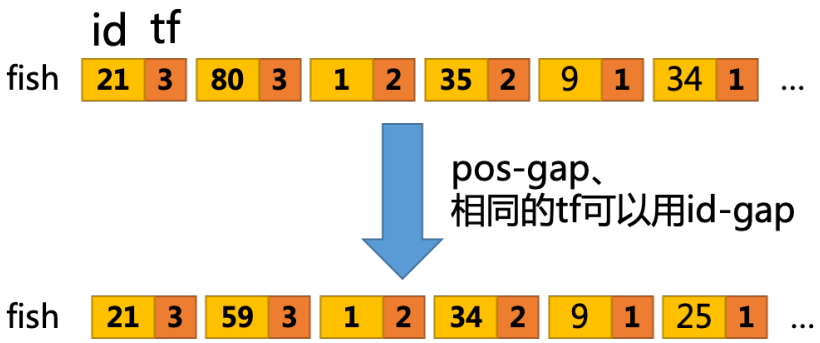

Figure 6: 有序的 ID、position 以 gap 的形式存储

这两种有损压缩方式可以极显著地压缩索引，但是本身会带来很大的信息丢失，而且很可能会导致在检索的时候无法找到本来相关性高的文章。因此，这两种有损的压缩方式需要慎用！

3.2 文章信息索引

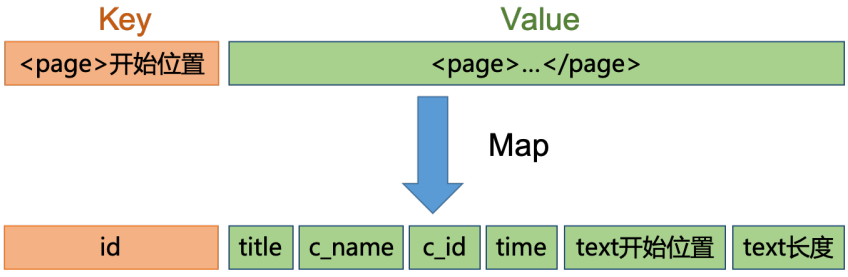

Figure 7: 文章信息索引的 Map 过程

如果在构建倒排索引的时候，还另外记录一些文章其他的信息，如标题、时间、贡献者、内容等，那么会产生数量巨大的数据冗余，而且对倒排索引的建立毫无帮助，并且会增加内存压力、延长索引建立时间。因此基于数据库的范式要求，如果想建立文章其他信息的索引，需要另外建立。

文章信息的索引很容易建立，如 Figure 7所示，只需要 Map Reduce 框架的 Map 阶段，把以 <page> 开始位置为 key、文章文本内容为 value，map 到以文章 ID 为 key，以标题 title、贡献人名字 c_name、贡献人编号 c_id、时间戳 time、文本开始位置、文本长度为 value，就可以了。通过文章 ID 可以索引到文章标题等其他信息，并可以通过文本开始位置、文本长度信息在 Wikimedia dump 文件中获得文本内容。

3.3 索引的索引

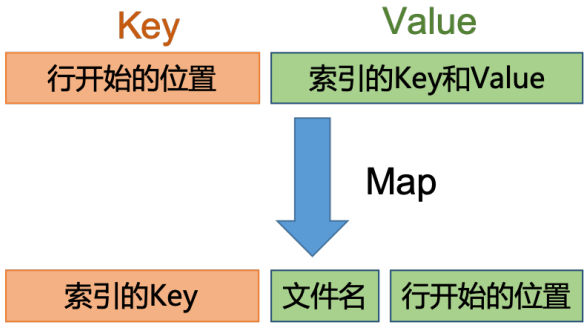

Figure 8: 索引上的索引的 Map 过程

构建好的倒排索引可能依然会很大，可能不能把整个索引放到内存中，即使放的下，也可能会使内存压力很大。那么可以在索引上建立索引，通过增加硬盘读取来降低内存的压力。

索引的索引就是要把索引的 key 映射到索引所在的文件和位置。那么通过索引的索引就可以找到 key 所在地方，也就可以获得 key 对应的 value。如 Figure 8所示，索引的索引建立也只需要 Map 阶段就可以了。Map 阶段以行开始的位置为 key，索引的 key 和 value 为 value 输入，这里直接采用 Map Reduce 框架的默认输入方式即可。Map 阶段以索引的 key 为键值，以文件名、文件中的位置为 value 输出，就可以构建好索引的索引了。

3.4 排序检索方法

排序检索就是分析查询项，根据倒排索引找到查询项中的单词对应的文章 ID、TF、position 信息，再通过给定的打分方式，对倒排索引建立的这些信息进行打分，对打分结果从高到低进行排序，输出高分的文章，就是检索的结果。

而打分的过程，就是要尽量使和查询项相关的文章获得更高分，而无关的文章获得更低分。打分方式实现了以下两种。

3.4.1 document-at-a-time 计分方式

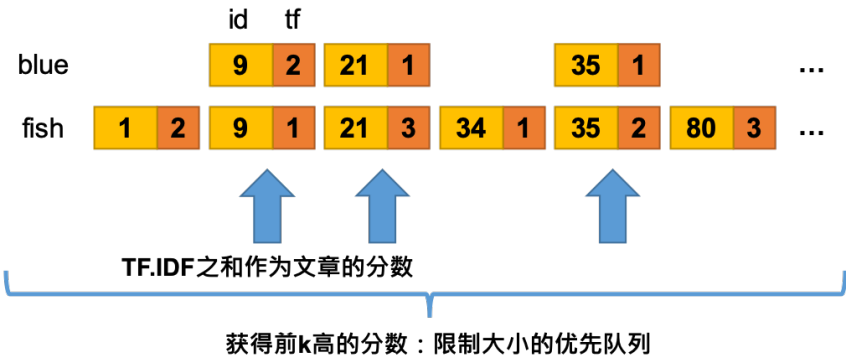

Figure 9: 应用在以 ID 排序的倒排索引上的 document-at-a-time 计分方式

检索的时候，首先要把查询项中的单词提取出来，根据倒排索引获得各个单词对应的文章列表。在文章列表以 ID 进行排序的情况下，可以使用 document-at-a-time 的计分方式。就是要同时顺序遍历所有的文章列表，找到在全部列表中都出现的文章，并对这些文章进行打分，每次产生一片文章的总分，如 Figure 9。由于这种方法需要顺序遍历文章列表，并且要求文章列表的 ID 是有序的，因此这种方式适用于以 ID 排序的倒排索引。项目中的以 ID 排序的倒排索引就是使用这个方法进行检索打分，

这种方式的缺点是要遍历整个文章列表。而且由于存储的是 ID 的 gap，而不是 ID，因此如果要采用 skipping 的方式去计分，还需要先遍历整个文章列表把 ID 还原出来，使得 skipping 的方式不可取。

3.4.2 query-at-a-time 计分方式

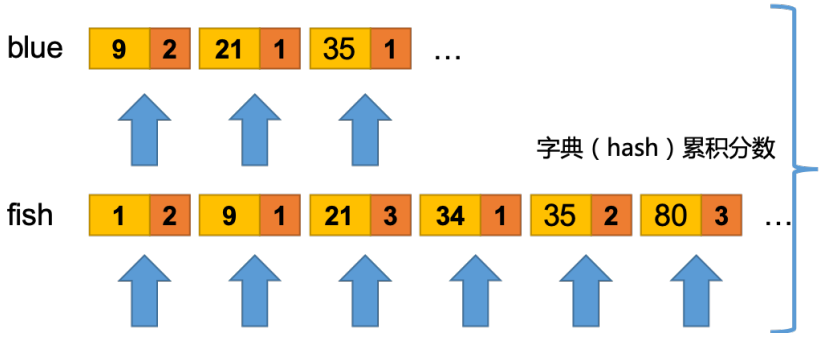

Figure 10: 应用在以 TF 排序的倒排索引上的 query-at-a-time 计分方式

相应地，在文章列表以 TF 为顺序排序的时候，可以采用 query-at-a-time 计分方式。也就是每次遍历一个单词对应的文章列表，并用 hash map 累积出现过的文章的分数。最后获得所有文章的分数，进行排序，就可以获得高分的检索结果，如 Figure 10所示。

这种方法的优点是，如果只需要 top-k 结果（通常都是这样的），可以使用大小为 k 的优先队列，可以降低内存的要求。而这种方法则只适用于以 TF 排序的倒排索引。项目中的以 TF 排序的倒排索引就是使用这个方法进行检索打分，

以上两种打分方式下，对一个单词对应的一篇文章的打分都是采用 TF.IDF 公式进行打分。而查询项整体对应一片文章的分数则是采用了查询项中的各个单词对应该片文章的 TF.IDF 分数之和。

3.5 检索网站实现

3.4中的排序检索方法都是使用 Python 实现的，而检索网站的后端也同样是用 Python 语言。通过 Flask 库搭建网站，并用 Materialize CSS 定义网站格式。网站主页有文本输入框，可以输入查询项，旁边的文本框用于输入 TF 或者 ID，表示在点击 Search 按钮之后会对应使用以 TF 排序的倒排索引或者用以 ID 排序的倒排索引进行查询。之后会返回查询分数 top 10 的文章结果列表，包括它们的文章标题、ID 以及对应的打分分数。各个文章标题还对应了该文章的 url 链接，点击后会进入文章的页面，可以看到文章标题、文章 ID、贡献者名字和编号、时间戳以及文章内容。

此外，还对查询项检索、文章内容检索过程进行计时，在页面中可以看到查询项检索、文章内容检索所需要的时间。

4 实验结果

4.1 数据处理

hadoop 提供的文件读入方式默认支持单行的读入，但是 Wikipedia dump[1]的文章信息在 <page>、</page> 之间的多行中，因此需要自定义读入方式，使得可以获得 <page>、</page>

[1]数据可以在https://dumps.wikimedia.org/enwiki/20191101/下载

Table 1: 基于 Map Reduce 实现的索引建立信息

项目	条目数量	大小	运行时间	索引上的索引大小
Wikimedia dump	19758736	70G	-	-
以 ID 排序的倒排索引	19099730	47G	1h	633M
以 TF, ID 排序的倒排索引	19099730	60G	1h	689M
有损的以 TF, ID 排序的倒排索引	19092140	7.8G	50min	661M
文章信息的索引	19758736	1.7G	15min	619M

之间的文章信息作为 Map 阶段的 value 输入，因此需要定义适合 Wikipedia dump 的 xml 文件的 TextInputFormat、createRecordReader 和 RecordReader。使默认的输入 value 从一行的内容变为 <page>、</page> 之间的内容。

这个 xml 文件读入方式需要在倒排索引、文章信息索引的建立中用到。

4.2 索引建立结果

如 Table 1所示，原文件的大小为 70G，而包含了 position 信息的以 ID 排序的倒排索引也仍然有 47G，所以是很有必要在索引上建立索引的。索引上的索引都是 600M 多，那么在只使用一种倒排索引的情况下，内容中只需要放倒排索引的索引、文章信息的索引的索引，也就是 1.3G 左右，内存压力还是不算大的。

倒排索引的建立还是需要约 1h 的。

4.3 查询结果

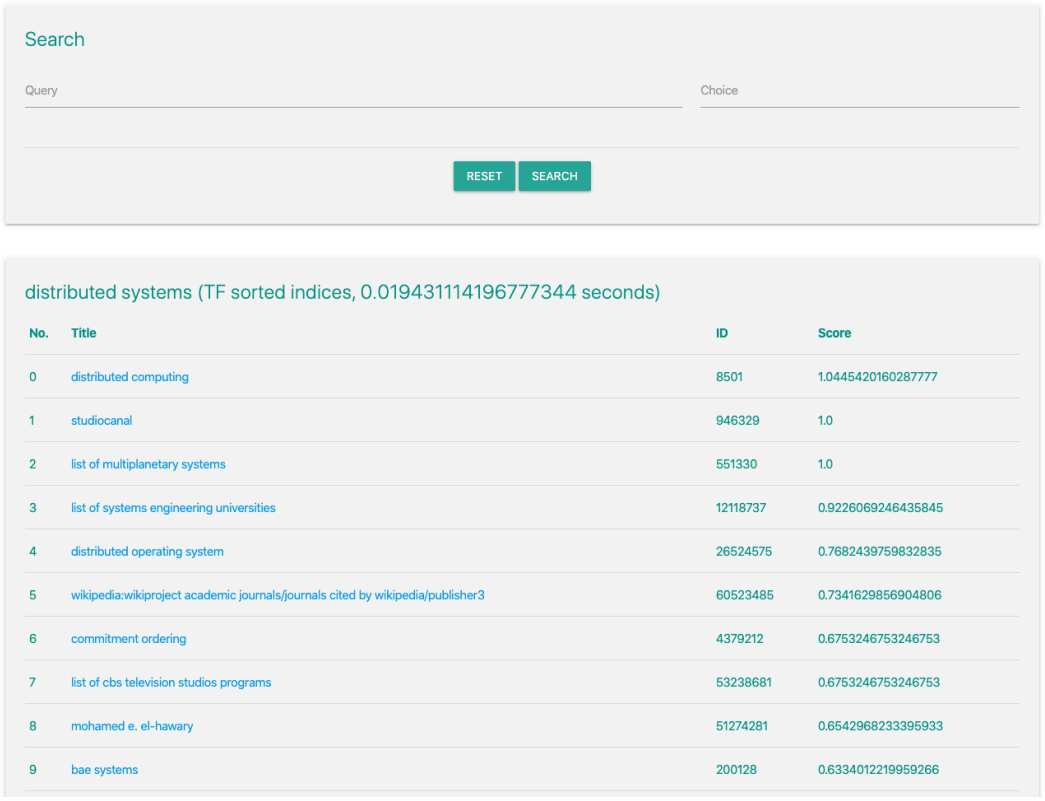

Figure 11: 查询结果列表

查询项检索'distributed systems' 的结果如 Figure 11所示。通过网站测试发现，基于 TF 排序的索引的检索一般在 0.01s 的级别，而基于 ID 排序的索引的检索一般在 1s 的级别，因此

基于 TF 排序的倒排索引的检索方法 query-at-a-time 要比基于 TF 的倒排排序的检索方法 document-at-a-time 要更快。而具体检索时间也与查询项中的单词本身和单词数量有关，单词本身越常见、单词数量越多，则检索时间越长。

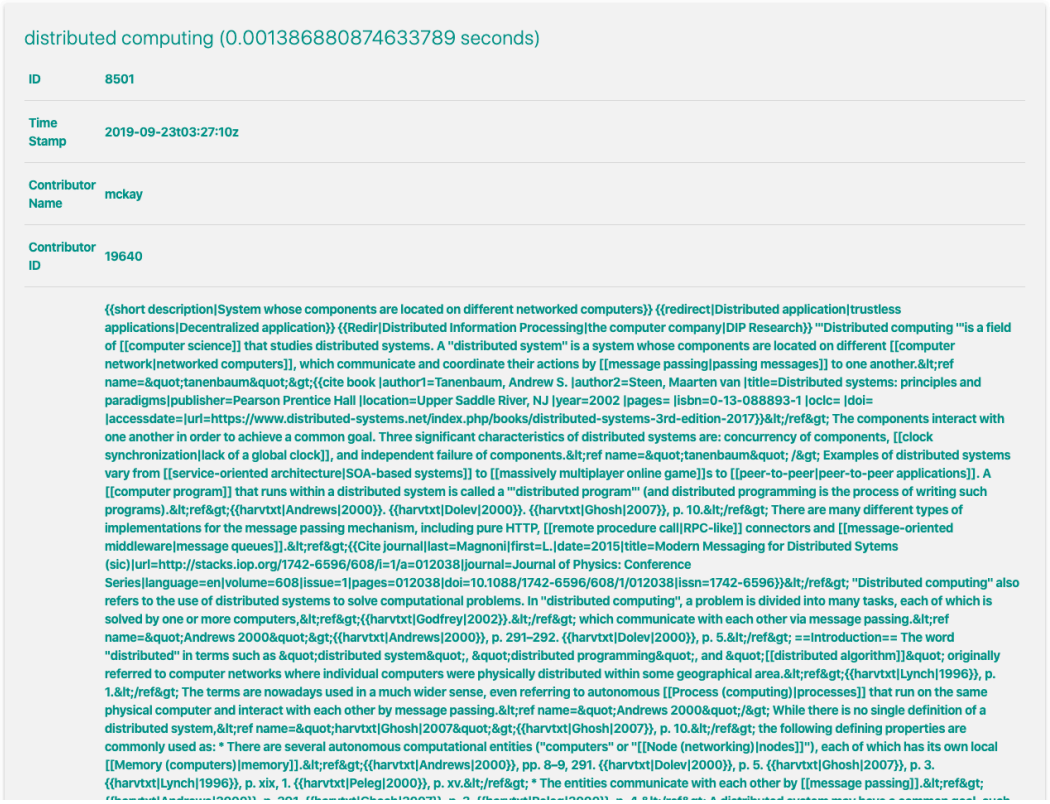

Figure 12: 查询文章信息

查询文章 ID 对应的文章信息和内容的结果如 Figure 12所示。通过网站测试发现，文章信息的检索一般在 0.001s 的级别，查询速度还是特别快的。

5 讨论

丢失了 position 信息 在数据处理的时候采用的是 bag of words 的理念，也就是只考虑了单词出现的次数，也没有考虑单词之间的相互位置关系，这样会对多个词的检索有致命的打击。

没有利用 position 信息 尽管倒排索引中记录了每个词的 position 信息，但是并没有使用过。而且，可以结合 position 信息，来获得词之间的相互位置关系，可以用来优化多个词的检索结果。

索引压缩方法的实践 项目中只利用 gap 的特点来编码有序的 ID 和有序的 position，进行无损压缩，而项目中尝试的有损压缩有限，没有探索其他的无损压缩方式。