应用 Elasticsearch 重构图书馆站内搜索引擎

陈俊杰　黄国凡

(厦门大学图书馆　福建厦门　361005)

摘　要:由于云搜索服务的灵活度和稳定性无法满足需求,厦门大学图书馆决定应用 Elasticsearch 重构站内搜索。通过环境部署、索引配置、数据导入、索引错误处理等步骤,厦门大学图书馆成功实现了基于 Elasticsearch 的站内搜索,且在功能、界面等方面有所改进。据此进一步提出,Elasticsearch 未来可应用于大数据环境,成为数字图书馆的私有云搜索引擎。

关键词:站内搜索;云服务;Elasticsearch;搜索引擎

中图分类号:G254.928　　**文献标识码**:A　　doi:10.3969/j.issn.1005-8095.2014.11.029

Using Elasticsearch to Rebuild Library's Intrawebsite Search Engine

Chen Junjie　HuangGuofan

(Xiamen University Library, Xiamen Fujian 361005)

Abstract: Ximen University Library decides to use Elasticsearch to rebuild intrawebsite search, because the flexibility and stability of cloud search service cannot keep up demand. Xiamen University Library successfully achieves Elasticsearch-based intrawebsite search by taking the steps of environment arrangement, index configuration, data importing, index error handling and so on, furthermore the function and the interface are improved to a certain degree. At last, the paper hereby points out that Elasticsearch can be applied in the environment of big data, and will become private cloud search engine of digital library.

Keywords: intrawebsite search; cloud service; Elasticsearch; search engine

0　引言

2013 年初,厦门大学图书馆构建了基于阿里云搜索服务的站内搜索引擎。阿里云搜索服务高质量、易扩展、可高度定制化的特点是厦门大学图书馆采用其构建站内搜索引擎的主要因素[1]。将近一年的使用体验,结合用户的反馈,笔者发现该搜索引擎存在两大问题:

(1)基于云服务的站内搜索引擎灵活度不够,无法调整排序算法及特定记录的权重。例如用户在站内搜索中最经常查找的是"数据库资源",那就有必要在数据库记录中提高这一类别在整体排名中的权重;最新发布的"公告"应该最优先推送给用户,但随着时间的推移,该公告信息的重要性逐渐下降,应使其在搜索结果的排名中处于次要位置。而阿里云搜索服务只提供针对记录级别的 boost(权重)值提升,如果要实现权重排序功能,只能定期由程序批量更新记录的 boost 值。

(2)云服务的稳定性令人担忧。云服务的最大问题在于,技术的后续发展严重依赖于厂商的市场策略。笔者迄今无法得到针对阿里云搜索的售后支持。由于用户数量较少,缺乏社区的支持,所以除了官方少量的文档及 SDK 范例,难以获得更多的资料。

如果这两个问题无法得到有效解决,随着图书馆网站内容的增长,站内搜索的效率和用户体验将难以提升。因此采用新的技术手段实现站内搜索显得必要和紧迫。近年来,开源搜索引擎不断发展,Lucene[2]、Sphinx[3]、Xapian[4]等代表性开源平台已成为站内搜索的新架构。经过比较和考察,笔者选择应用 Elasticsearch[5]重构图书馆的站内搜索引擎。

1　Elasticsearch 简介

Elasticsearch(以下简称 ES)由美国 Elasticsearch BV 公司开发,是一个开源的分布式实时搜索与分析引擎,支持云服务。它是基于 Lucene 的类库创建的,提供了全文搜索、多语言支持、专门的查询语言、支持地理位置服务、基于上下文的搜索建议、自动完成以及搜索片段(Snippet)的能力。它致力于使开发者能够使用尽量少的开发成本快速实现一个功能完善的检索系统,主要特点如下:

(1)提供完整的 RESTful[6] API。所有的操作,包括搜索、统计分析、管理、监控都能够使用发送基于 JSON[7]的 http 请求实现。

(2)准实时。这是 ES 优于传统搜索引擎解决方

收稿日期:2014-04-23

作者简介:陈俊杰(1983-),男,硕士,馆员,主要研究方向为数字图书馆相关前沿开发技术,搜索引擎开发、数据挖掘等;黄国凡(1971-),男,本科,副研究馆员,主要研究方向为数字图书馆、新媒体服务等。

案,如 Solr[8]的一个重要特性。对于索引更新频繁的系统,新增索引的即时可用性能够得到保障。

(3)高可用(Highly-available)、高伸缩(Scalable)、分布式。这三个特性都是源于 ES 简易但强大的分布式支持。在 ES 中,索引采用分片(shards)存储,同时可设置冗余副本(replica)的数量。任何一个新增的节点都能够快速、自动的加入 ES 集群,而且整个过程对于用户和开发者是完全透明的。当某一个节点故障时,其余的节点能够保障检索服务的持续稳定运行。

(4)强大的社区。越来越多的系统开始采用 ES 作为检索系统的解决方案,如最强大的代码管理网站 Github[9]以及国内的唯品会等。随着使用者的日益增多,ES 开发团队自建的文档以及使用者的经验共享资源也越来越丰富,极大地降低了开发者的使用门槛。

2　应用 ES 实现站内搜索

ES 相比于其他搜索引擎解决方案的一大优点就是高可用性,它为用户提供了完善的默认设置,直接启动服务,一个完整功能的搜索引擎就已经处于可用状态,即所谓的“开箱即用”(out of box)。

/bin/elasticsearch -f # 程序解压后直接运行该命令即可启动完整功能的搜索引擎

这种易用性甚至扩展到数据的索引过程,用户将 JSON 格式的数据提交给服务器,ES 可自动识别数据的类型并进行相应的索引操作,如需进行个性化定制,ES 也提供了灵活便利的配置方法。笔者将在后续部分详述如何定制 ES,实现厦门大学图书馆的站内搜索。

2.1　部署环境

ES 采用 Java 开发,原生支持跨平台部署使用,任何服务器只需安装 Java 运行环境即可部署 ES。

java -version # 安装 Elasticsearch 前确保该命令可执行 Java 平台程序需要根据应用需求对 JVM 的各个参数进行调优,但默认情况下 ES 已经预先为 JVM 进行了优化配置,除非必要,用户无需关心 JVM 的设置就可以马上使用。笔者在实现“厦门大学图书馆站内搜索”时并未修改这部分的设置就获得了很好的性能。

2.2　索引配置

建立索引是构建搜索引擎的基础,索引的设计及策略决定了搜索引擎的索引效率、检索效率、召回率、准确率及索引膨胀率。ES 的核心技术是基于 Lucene 开发的,因此其索引过程与 Lucene 相似,但也有其独有的特点。索引的基本单元是文档(document),索引机制包含以下几个要素:类型(type)、解析器(analyzer)、存储策略(store),解析器又包含分词器(tokenizer)及预处理过滤器(token filter)。

(1)“类型”决定字段的基础处理方式。以内容为中文,类型设置为 string 的字段为例,默认的解析器 standard analyzer 会将文本中每个字作为一个单元进行解析。又如 integer 类型的字段会解析为数字,默认支持检索过程的过滤及排序。

(2)“解析器”是索引机制的核心,决定系统在索引写入及检索过程中如何处理字段中的内容。它包含分词器及预处理过滤器,分词器将原始字符串拆分成写入索引的最小基本单元,如将英文句子分成一个个单词;而预处理过滤器则对这些最小基本单元进行预处理,如 stopwords 的预处理过滤器会将字符串中的停用词去掉,不进行索引。

(3)将“存储”设置为 false,则系统不会对该字段进行存储。部分只用于搜索,无需显示在结果中的字段就采用 false 的设置,这样能够减小索引所占空间。以“厦门大学图书馆站内搜索”为例,记录的具体描述字段 body 包含大段的文本,我们只需将其用于搜索,而不会在搜索结果中将其完整显示给用户,因此就将“store”设置为 false。

(4)“分词”在中文检索中至关重要,分词的准确率与搜索的准确率成高度正向相关。ES 默认提供的以字为单位的分词明显无法符合厦门大学图书馆站内搜索的需求。用户进行站内搜索时输入的字符串可能是中文或英文,因此需要解决两个问题:普通中文分词以及英文词干的提取。例如字段中包含单词“pictures”,则用户输入“picutre”或者“pictures”都希望能命中结果,为此需要自定义整合中文分词及英文词干提取的解析器。中文分词器有 ES 官方提供的 smartcn[10]以及第三方开发的 ik[11]及 mmseg[12]。经测试,笔者采用了在易用性、分词效果方面表现最好的分词器插件 ik。词干提取方面,官方提供的解析器是 token filter。最终的整合方法是在 ES 的配置文件 elasticsearch.yml 文件中添加如下设置:

```
index:
    analysis:
        analyzer:
            ik:
                alias:[ik_analyzer]
                type:org.elasticsearch.index.analysis.IkAnalyzerProvider
            ik_max_word:
                type:ik
                user_smart: false
            ik_smart:
                type: ik
```

```
                user_smart: true
            ik_stem: # 自定义的解析器，整合英文词干提取及 ik 中文分词器
                filter: [myStemmer]
                tokenizer: ik_tokenizer
        tokenizer :
            ik_tokenizer : # 定义 ik 作为分词器
                type : ik
        filter :
            myStemmer : # 自定义英文词干提取的预处理过滤器
                type : stemmer
                name : english

}
```

2.3 数据导入

ES 支持逐条记录的索引写入，也支持批量写入。两者的差别在于处理的效率及索引写入时的系统延迟。每次提交写入索引的请求所耗费的时间包括 http 连接建立的时间和索引数据的传输时间，而批量写入能够节省大量的 http 连接建立时间。默认每次写入索引后一定时间周期内会执行 Refresh 以供检索，默认周期是 1s，而 bulk 的批量导入能够极大的降低 Refresh 的次数，从而降低索引写入对搜索的影响。

ES 的批量索引通过 Bulk API 来实现：

```
client.bulk body: [
    { index: { _index: 'znss',_type: 'item',_id: 1,data: { title: 'foo' } } },
    { index: { _index: 'znss',_type: 'item',_id: 2,data: { title: 'foo' } } },
    { index: { _index: 'znss',_type: 'item',_id: 3,data: { title: 'foo' } } },
]
```

即将所有请求封装到一个 body 中，然后调用 bulk 接口提交请求。每个 bulk 请求中包含记录的数量并无限定，具体取决于系统的硬件配置以及网络情况等。

笔者比较了逐条写入索引及各个数量的 bulk 写入的效率差别，见表 1。

表 1 几种数据导入方式的效率比较

数据导入方式	记录数	时间	平均导入速度
逐条	50793	1683.218680	30.180/s
bulk: 1000	50793	543.904086	93.541/s
bulk: 1500	495000	6321.597719	78.29/s
bulk: 2000	74000	919.905391	80.43/s

从以上实验数据可以得出两个结论：一是单就索引效率而言，bulk 的方式明显高得多；二是每个 bulk 请求包含的记录数量并非越高越好，而是取决于服务器的配置。

2.4 索引错误处理

大量数据写入索引时可能出现错误，原因有二：

① 网络错误：索引数据在 http 传输过程中由于网络不稳定等原因中断。

② 解析错误：上文提及的模式中定义了各个字段的类型，写入索引的数据必须符合相应字段类型。以最严格的 Date 类型为例，其默认的格式规范包括“YYYY-mm-DD”、“hh:MM:ss”等几种，原始数据中出现规范之外的字符就会触发解析错误，导致整条记录索引失败。

为了保证数据的完整性，索引错误处理机制不可或缺。具体流程如下：

① 调查已有数据各个字段的内容情况，据此在模式映射时多添加一些自定义的格式，如添加“xxxx 年 xx 月 xx 日”的规范到 Date 类型字段；

② 提交数据前的数据清理(Data Cleaning)；

③ 为保证原始数据在索引之后不丢失，设计专用字段用于存储不合规范的字段内容；

④ 索引过程中将错误信息详细记录到特定位置，便于后期查看，重做索引；

⑤ 管理员查看错误信息后重做索引。

上述第 3 步处理方式较为特殊，实现方法是利用 ES 提供的 multi field 字段类型。该字段类型能够实现单字段中存储多字段的复杂信息。

以“发布日期”字段为例，正常的模式映射如下：

```
mappings: {
    item: {
        properties: {
            publish_date: {type: 'date',store: 'yes'}
}}}
```

笔者的处理方式是利用 multi field 类型的特性，在 publish_date 字段中设置两个字段：与原字段同名的 publish_date，以及 publish_date.untouched 字段。在检查原数据符合 Date 规范时写入 publish_date 字段，不符合时则 publish_date 留空，写入 publish_date.untouched 字段。

```
mappings: {
    item: {
        properties: {
            publish_date: {type: 'multi_field',fields:
{
                publish_date:   {type: 'date',store:
```

```
´yes´},
                    untouched:  {type: ´string´,in-
dex: ´not_analyzed´}
                }}}}}
```

2.5　搜索实现

ES 提供完善的 RESTful API，任何形式的搜索都可通过发送 http 请求到相应 API 实现。另外 ES 本身也提供覆盖主流开发语言的客户端 SDK，用于与搜索服务器便捷地实现交互。

针对请求的内容,ES 定义了专用的基于 JSON 的检索领域语言（Query DSL），用于复杂请求的包装。以搜索引擎最为常用的针对多个字段进行检索，同时返回分面信息、高亮信息的请求为例：

```
curl -XGET http://x.x.x.x:9200/znss/_search -d
´{
    "explain":true,
    :query => {"query_string" :
                    {"fields" : ["title^2","body","
author","source"],
                    "query" : "xxx"}},
    size: size,
    from: (page-1)*size,
    facets: {
        type_id: {terms: {field: ´type_id´}},
        cat_id: {terms: {field: ´cat_id´}}
    },
    "highlight" => {
        "pre_tags" => [
            "<em>"
    ],
    "post_tags" => [
        "</em>"
    ],
    "fields" => {
        "title" => {},
        "body" => {}
    }
}´
```

以上代码用 JSON 封装请求,用 curl 工具[13]发送 GET 请求到搜索服务器。请求的内容包括:explain 返回每个结果的排序权重计算结果值,query 指定多个字段进行检索,并将 title 字段的 boost 设置为默认值的两倍,因为从经验而言,title 的重要性显然要高于其他字段。对于每个字段的权重能够在索引写入及搜索时进行设置,如果没有设置,则每个字段默认权重都为 1。在索引写入时就设置字段权重,能够提高搜索效率,但是后期无法进行修改。因此,出于动态调整搜索排序算法的灵活性，在具体搜索时才设置字段权重。上述例子只是最简单的动态调整排序算法的应用，通过定制更为复杂的公式还可实现根据时间推移调整排序。最后,facet 指定对 type_id、cat_id 字段进行分段处理,highlight 设置在 title 及 body 字段中高亮显示匹配的字符串。

2.6　交互设计

ES 本身是一个搜索引擎解决方案,负责后台索引的管理及提供检索服务，前台的用户交互部分则需自行设计开发。

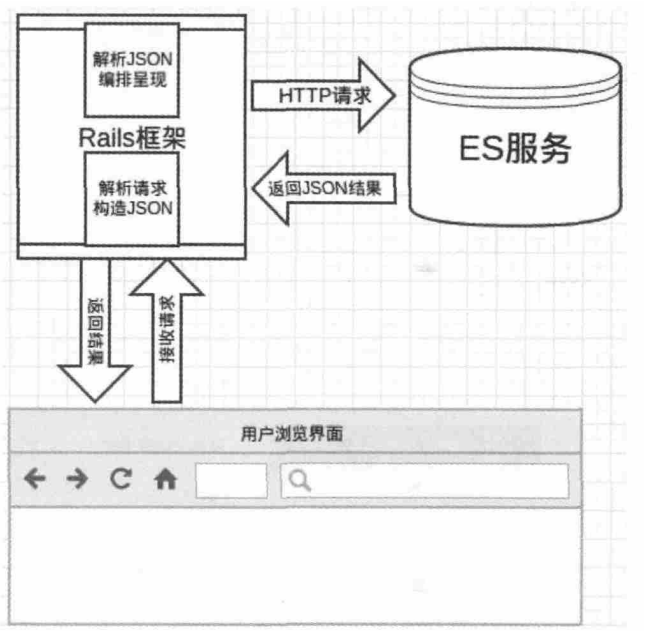

图 1　应用 ES 实现站内搜索的架构

如图 1 所示，笔者在前端部分采用 Ruby on Rails[14]框架(以下简称 Rails)进行开发。Rails 是由 Ruby 语言开发的开源 Web 应用框架，主要作用是将 JSON 对象以 HTTP POST 的方法发送到服务器相应的 URL。ES 为 Rails 提供了封装完善的 SDK,开发者直接调用 SDK 中的相应方法即可实现搜索,无需关注底层 HTTP 请求发送相关事宜。因此,使用 Rails 响应用户请求、与 ES 服务器进行交互的开发过程极为便捷。

```
def search_with_facet q,page,size,sort_field,filter_id
host = "http://x.x.x.x:9200"
@client = Elasticsearch::Client.new host: host,log: true
body_json = {
    "explain" => true,
:query => {"query_string" =>
  {"fields" => ["title^2","creator","subject","descrip-
tion","contributor","publisher","type"],
```

```
"query" => "\"#{q}\""}},
size: size,
from: (page-1)*size,
facets: {
  publisher: {terms: {field: 'publisher.untouched'}},
subject: {terms: {field: 'subject.untouched'}},
creator: {terms: {field: 'creator.untouched'}},
contributor: {terms: {field: 'contributor.untouched'}},
rights: {terms: {field: 'rights'}},
histol: {
  date_histogram: {
    field: 'harvest_time',
  interval: 'minute'
  }
  }
},
"highlight" => {
"pre_tags" => [
  "<em>"
],
"post_tags" => [
"</em>"
],
"fields" => {
  "title" => {},
"body" => {}
 }
}}
if ! sort_field.nil?
body_json[:sort]= {sort_field => {"order" => "desc"}}
end
if ! filter_id.nil?
  body_json [:filter]= {"term" => {"cat_id" => filter_id}}
end
  @client.search index:'oai_ik_stem_2',body:body_json
end
```

以上代码就是一个完整的搜索请求的实现过程。整个流程首先是调用 Elasticsearch::Client.new 方法建立与 ES 服务器的连接,之后根据前面介绍的方法构造 JSON 对象 body_json, 最后调用 @client.search 方法,指定对应的索引名称,把 body_json 发送出去,最终获得搜索结果,如图 2 所示。

图 2　站内搜索字符串"时间"的页面效果

3　应用效果评估

从 2013 年 12 月至今的实际使用和测试来看, ES 很好地满足了站内搜索的需求。看似复杂的排序算法动态调整在 ES 中只需根据文档在检索时构造

简单的公式即可便捷快速地实现。新版站内搜索上线以来,从服务器日志中未发现由于 ES 服务导致的错误,服务十分稳定。目前站内搜索的数据量为 2125 条,每次 ES 检索请求的响应时间都在 50 毫秒内,有时甚至小于 10 毫秒。

4　ES 在数字图书馆的应用前景

笔者应用 ES 重构站内搜索的原因除了云服务的不灵活以及稳定性堪忧之外,主要是希望借助站内搜索的应用,探讨 ES 作为私有云搜索引擎的可行性。不管在部署还是数据交互方面,ES 均表现良好,适合作为数字图书馆私有云搜索平台,为所有应用系统提供搜索服务。厦门大学图书馆采用开源工具定期自动抓取网络中开放获取的数据,以这些数据为基础构建了一个开放获取数据仓储平台,目前该平台的检索系统亦采用 ES 实现。截止到 2014 年 2 月 10 日,该平台抓取的元数据总量约 2500 万,每条记录字段数为 18 个,占磁盘空间 17.99G,且数据总量还在增长中。作为"为大数据而生"的搜索引擎,ES 的应用前景相当广阔。未来,笔者将继续考察 ES 的性能,拓展其在数字图书馆的应用。

参考文献

[1]　王爽,陈俊,杰肖铮,等. 应用阿里云搜索服务构建图书馆站内搜索引擎 [J]. 现代图书情报技术,2013 (6):85-89.

[2]　王兆宇,乐嘉锦. 基于 Lucene 的个性化站内搜索引擎的研究[J]. 计算机应用与软件,2011(12):188-190,223.

[3]　刘清明,彭宇扬,彭自成. 基于 Sphinx 的 Web 站内搜索引擎的设计与实现 [J]. 微计算机信息,2010 (15):116-118.

[4]　孙文慧,魏幼平. 基于 Xapian 和 PHP 的高性能站内搜索系统方案设计[J]. 计算机与现代化,2012(4):76-78.

[5]　Rafal Kuc,Marek Rogozinski. Elasticsearch Server [M].Birmingham:Packt Publishing Ltd.,2013.

[6]　Leonard Richardson,Sam Ruby. RESTful Web Services[M]. Sebastopol:O'Reilly Media,Inc.,2007.

[7]　Deborah Nolan,Duncan Temple Lang. XML and Web Technologies for Data Sciences with R [M]. New York:Springer Science+Business Media,2014.

[8]　David Smiley,Eric Pugh.Solr 1.4 Enterprise Search Server[M]. Birmingham:Packt Publishing Ltd.,2009.

[9]　A Whole New Code Search [EB/OL]. [2014-03-14]. https://github.com/blog/1381 -a -whole -new -code -search.

[10]　Smart Chinese Analysis for Elasticsearch[EB/OL]. [2014 -03 -14]. https://github.com/elasticsearch/elasticsearch-analysis-smartcn.

[11]　IK Analysis for ElasticSearch[EB/OL]. [2014-03-14]. https://github.com/medcl/elasticsearch-analysis-ik.

[12]　Mmseg Analysis for ElasticSearch[EB/OL]. [2014-03-14]. https://github.com/medcl/elasticsearch-analysis-mmseg.

[13]　cURL[EB/OL]. [2014-03-14].http://curl.haxx.se/.

[14]　Ruby on Rails Guides (v4.0.3) [EB/OL]. [2014-03-14]. http://guides.rubyonrails.org/.

(上接第 113 页)

存储设备)而不影响正常业务运行。

(5) 服务器整合。VMware 能够帮助用户进行服务器整合、创建可升级的开发/测试环境,以及实现商业连续性的策略等。在同一台物理服务器上安装多个虚拟服务器操作系统,无论是出于测试还是使用的目的,都能够以一种很经济的方式达到商业扩展或者增加服务器资源的目的 [4]。例如在医院的 HIS、PACS、LIS 等业务系统数据库,我们可以搭建一套甚至多套的生产系统测试环境、作为测试或者数据的备份。

(6)对灾难恢复的支持。服务器虚拟化正迅速成为许多组织灾难恢复(DR)策略中的一个关键组成部分,因为借助虚拟化技术的某些特殊功能,可以精简灾难恢复的过程。将虚拟机整齐地封装到独立的存储单元后,可以简化灾难恢复时对虚拟机的处理。SAN 复制软件可以将虚拟机复制到灾难恢复站点,由于虚拟机的硬件无关和封装特性,复制虚拟机比复制传统物理机要快得多。例如一些医院单独运行的小型业务系统,我们在发生故障时、可以用最短的时间进行业务系统的修复并恢复生产。

5　结语

结合医院的业务系统使用虚拟化技术带给我们安全、稳定、便捷的信息化平台,降低设备成本与节能、提高资源管理效率、灵活扩展、充分利用现有设备和专业软件、提供弹性的计算能力、维护方式简便、业务系统快速部署。我们以整合资源、节约成本的目的开始,过程中不断的完善各个环节的单点故障,实现整套系统的高度整合及高可用性。

参考文献

[1]　李先锋,王凯芸,吕强,等. 三甲医院虚拟化技术的研究与实践[J]. 中国医院,2012,16(2):12-14.

[2]　房秉毅,张云勇,陈清金,等. 云计算网络虚拟化技术[J]. 信息通讯技术,2011,5(1):50-53.

[3]　潘欣. 服务器虚拟化技术在医院信息系统中的应用[J]. 中国卫生产业,2013(1):124-125.

[4]　马秀芳,李红岩. 计算机虚拟化技术浅析[J]. 电脑知识与技术,2010,6(33):9408-9409,9412.