第四章 绪论

本毕业论文的题目为学术搜索引擎大规模查询系统的建立与优化，论文主要讨论如何为学术搜

索网站建立一个高效稳定的大规模查询系统。在此基础上，论文还讨论了如何对系统进行优化，功

能扩展与分布式部署，同时研究了如何设计根据搜索条件动态生成知识层级图的算法。

论文中所有的实验，系统部署与验证都在学术搜索平台 ACEMAP 上进行。目前绝大多数学术网

站也都提供了功能完善的论文搜索功能，因而在 acemap 中部署搜索平台的工作也是必要的。如何设

计并实现一个大规模的查询系统是一个较为复杂的工程问题，在技术细节上也会有很多难点与挑战。

在这一部分的工作中，主要的工作可以列举如下：

1. 系统架构与索引结构的设计

2. 大规模数据的高效导入

3. 搜索后台与前端的接口设计

4. 查询系统的分布式部署

本论文在第二到五章依次讨论了这些问题，这构成了本篇论文的主体部分。事实上，本文也是

一个偏设计类的论文。在研究的过程中，本人阅读了大量的官方与非官方的著作与相关文档，但由

于学术搜索这一应用较为特殊，数据库中总条目数很多且数据结构复杂，这导致参考文档只起到了

极为有限的参考作用。在大部分的成果中，本人的工作内容都是自己设计，实现并测试优化的。因

此，这项建立学术搜索平台的工作，不仅在设计上具有原创性和完整性，也针对遇到的棘手问题有

独到的解决方案，这也坚定了我将自己的工作整理写作成论文的信念。

在第六章中，论文讨论了如何在搜索结果的基础上建立层级化知识图。该功能的构想基于如下

考量：用户使用学术网站进行查询的时候，需要的除本领域的相关知识，还可能需要一些跨领域的

知识。在用户输入关键词进行查询的时候，可能只重点关注搜索结果的前几条，而隐藏在大量搜索

结果之中的知识信息则无法很好的被用户知晓。论文希望给出一种算法，可以基于查询结果自动生

成知识脉络图，挖掘出和用户搜索关键词相关的知识信息。例如对于 energy saving 这一输入，系统

能在环境科学、经济学、电气、化学、数学等领域梳理出相关研究的层级状知识图，以帮助用户更

好地确定与了解自己的研究方向。

第 21 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

第五章 学术搜索引擎查询系统架构与索引结构的设计

5.1 工作平台

查询系统采用的软硬件工作平台如下：

5.1.1 硬件部分

本论文讨论的查询系统的构建方式分为两个阶段。论文的第 2-4 章中着重讨论了在单台服务器

上独立部署服务器的相关内容。在第五章中讨论了包含两台服务器的分布式集群的建立方法。

在单机版本中使用的服务器硬件配置如下：

• CPU: 64 位 Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz \* 32

• 内存大小：约 128GB

• 硬盘空间：约 800TB

在分布式服务器中，除了用到了上述服务器之外，还使用了一台备用服务器用以建立集群。该服务

器的硬件配置如下：

• CPU: 64 位 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz \* 40

• 内存大小：约 128GB

• 硬盘空间：约 4TB

论文中所有功能的实现与验证都基于该硬件配置。由于查询系统索引规模较大，在单台服务器中索

引总大小约为 60G，分布式服务器中总大小约为 120G，因此将该系统移植到配置较低的计算机时可

能会出现问题。

5.1.2 软件部分

服务器的操作系统为 Ubuntu 14.04.1，Linux 内核版本号为 3.19.0-25，本论文实验部分对操作系

统版本无严格要求。同时，论文采用了Solr6.5.0作为搜索服务器的实现方案。Solr是一个基于Lucene

编写的开源搜索平台，由于查询系统建立与优化的重点并不在从倒排索引开始的底层实现上，因此

论文全部基于该平台进行系统的实现与部署。

在分布式集群中，查询系统采用了 Zookeeper 作为多个 Solr 内核的管理平台，该平台可以自动

管理内核的协调工作，包括配置文件统一管理，任务队列，计算资源分配，主服务器选举，宕机处

理等。

5.2 系统架构

查询系统分为两个部分，一部分在网站后台 PHP 代码中实现，主要控制用户输入的处理与分析，

并将分析后的结果传给搜索服务器后台；一部分在 Solr 平台上实现，主要负责文档的处理，索引的

建立与请求的标记解析。这一阶段我们暂时只考虑单台服务器架构下的系统架构。单台服务器上的

系统架构见表（7–1）。

第 23 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

网站视图 网站控制

查询预处理

查询解析

数据库

文档导入处理

文档分析

索引服务器

图 5–1 单服务器系统架构

Fig 5–1 System Architecture (Singled)

该架构各部分的作用如下：

• 网站视图：负责网页的显示效果与用户交互。显示效果包括搜索结果展示、统计结果展示、关

键词高亮效果等；用户交互包括查询输入框、翻页功能等。

• 网站控制：负责用户输入的处理与网站实际功能的实现。包括查询预处理，网站与搜索服务

器的通信机制与结果处理等。

• 查询预处理：该模块为网站控制的一部分。负责对用户输入的预处理，过滤特殊符号并使之

变为 html 格式的编码。

• 查询解析：该模块为查询平台的一部分。其目前只对英文内容进行处理，包括过滤停用词，大

小写统一，过滤敏感词，单词词根化等。

• 数据库：索引内容的数据源，包含了全部超过 1.2 亿条论文的信息。所有信息被保存在 7 张表

中，需要通过数据库的连接得到一篇论文的全部信息。

• 文档导入处理：该模块也是查询平台的一部分。其决定要如何从数据库中取出信息和需要取

出数据库的哪些信息。

• 文档分析：该模块和查询解析模块的功能类似。负责对文档部分字段（例如标题字段）的处

理，同样地，包括过滤停用词，大小写统一，过滤敏感词，单词词根化等步骤。

• 索引服务器：数据库中的数据最终被索引，以索引文件的形式保存在索引服务器中。同时，用

户的查询请求也在索引服务器中被处理。同时，索引服务器还承担了关键词高亮，结果统计

等扩展功能的工作。

5.3 平台配置

在设计完系统的整体架构后，需要对平台进行合适的配置，再能进行之后的索引结构设计与文档

导入的操作。在solr平台中，主要需要配置的是solrconfig.xml文件，它管理了平台的大部分工作配置，

包括启动时加载的模块，针对不同的请求的对应逻辑与系统备份、系统维护的操作等等。在一个刚刚

下载好的solr服务器中，该配置文件里已经有了一些最基本的配置，我们需要通过添加一些配置以实

现我们需要的功能。我们选择和本课题目标最为接近的安装包自带的 data\_driven\_schema\_configs 文

第 24 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

件夹下的 solrconfig.xml 文件的基础上进行修改。需要进行修改的地方如下：(参考文献：solrexample)

1. 由于文档导入涉及到数据库操作，因此要在软件启动时声明加载链接 mysql 数据库相关的.jar

包与数据导入处理相关的.jar 包。在 solrconfig.xml 中添加以下内容：

代码 5.1 solrconfig.xml 改动 1

<lib dir=${solr.install.dir:../../../..}/dist/ regex=mysql-connector-java-.\*j̇ar />

<lib dir=${solr.install.dir:../../../..}/contrib/dataimporthandler/lib/ regex=

.\*j̇ar />

<lib dir=${solr.install.dir:../../../..}/dist/ regex=solr-dataimporthandler-.\*j̇ar

/>

<lib dir=${solr.install.dir:../../../..}/contrib/dataimporthandler-extras/lib/

regex=.\*j̇ar />

2. 需要配置数据导入使用的包和默认采用的配置文件。对于以 HTML 参数形式传入的请求，都

可以在 solrconfig.xml 中进行处理配置。要实现此处的配置，需要在文件中添加以下内容：

代码 5.2 solrconfig.xml 改动 2

<requestHandler name=/dataimport class=

org.apache.solr.handler.dataimport.DataImportHandler>

<lst name=defaults>

<str name=config>db-data-config.xml</str>

<str name=clean>false</str>

</lst>

</requestHandler>

此处意为，当HTML请求传入的第一个参数为dataimport时，使用solr中的DataImportHandler

包对其进行处理。当不指定额外参数的时候，文档导入的配置文件读取文件名为 db-data-

config.xml 的文件，且进行后一次导入时默认不清空前一次导入的索引内容。

3. 对于大部分请求的默认域，系统默认配置为查询 \_text\_ 字段。由于本平台对默认查询字段有

特殊的要求，因此用 \_entext\_ 字段代替 \_text\_ 字段。此处，entext 意为 exglish text。关于该字

段的具体说明在下一个章节中有详细解释。我们需要修改配置文件中以下对应部分的 text 为

entext。

代码 5.3 solrconfig.xml 改动 3

<initParams path=/update/\*\*,/query,/select,/tvrh,/elevate,/spell,/browse>

<lst name=defaults>

<str name=df>\\_text\\_</str>

</lst>

</initParams>

4. 在搜索引擎的扩展功能，即代码高亮功能与结果统计功能中，也需要对此处的配置文件进行

相应的修改。在配置文件中，我们需要修改配置文件中处理 select 请求的部分，打开默认的

高亮与统计功能的接口开关。

第 25 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

代码 5.4 solrconfig.xml 改动 4

<requestHandler name=/select class=solr.SearchHandler>

<lst name=defaults>

<str name=echoParams>explicit</str>

<int name=rows>10</int>

<bool name=hl>true</bool>

<str name=hl.fl>OriginalVenueName OriginalPaperTitle</str>

<str name=hl.simple.pre>&lt;font color=#196600&gt;</str>

<str name=hl.simple.post>&lt;/font&gt;</str>

<bool name=facet>true</bool>

<str name=facet.field>PaperPublishYear</str>

<str name=facet.field>KeywordID</str>

<str name=facet.field>AuthorID</str>

<str name=facet.field>ConferenceSeriesID</str>

<str name=facet.field>JournalID</str>

</lst>

</requestHandler>

其中 hl 为高亮功能，hl.simple.pre 为希望设计的被高亮的字段的前缀，hl.simple.post 为高亮

字段后缀。由于最后高亮效果要在 html 页面中呈现，因此此处用了 html 格式的更改样式代

码。facet 为统计功能，facet.field 声明了需要被统计的域有哪些。

5.4 索引结构

本段内容描述了搜索平台中索引是如何构建的。索引的结构包括索引中需要包含哪些字段，字

段是否需要被索引 (index)，字段是否需要被保存 (store) 等。被索引的字段表示我们可以通过请求查

询该字段的内容，被保存的字段表示返回的结果中我们可以看到这部分的完整内容。考虑到既不需

要索引也不需要保存的字段完全不需要被索引，而在本学术搜索平台的实际应用中，并不存在只需

要索引且不需要保存的字段，因此，该索引中所有的字段都是需要被保存的。

本项目中需要索引的字段共有 13 个。其分别为：作者 ID、作者姓名、论文作者顺序、会议 ID、

会议简称、研究领域、期刊 ID、关键词 ID、关键词、论文标题、论文发表位置、论文出版年份和论

文被引用数。其中，对于每一篇文章，作者 ID 与姓名、作者顺序、关键词 ID 与名称这 5 个字段是

多值的（以列表表示），其余所有字段均是单值的；期刊 ID 和会议 ID 对于每一篇文章至多只有一项

不为空；论文出版年份以数的形式表示（这代表该字段可以在搜索条件中以区间作限制），其余所有

字段均以字符串的形式做表示。

这 13 个字段都是被保存 (STORE) 的字段，它们虽然有一些并不会在网页中被直接显示，但是

有可能会在统计功能的图表绘图中被使用（例如会议 ID，期刊 ID，关键词 ID 等）。同时，这 13 个

字段中只有 5 个字段是被索引的，也就是论文标题，作者姓名，发表处名称，出版年份和发表会议

简称。也就是用户只能通过这 5 个字段查询想要的内容，通过关键词搜索论文，是不能搜索到关键

词被包含于这 5 个字段之外的字段的论文的。在这 5 个被索引字段中，作者姓名、出版年份和发表

会议是不会被文档分析模块处理的。也就是单词词根化等操作不会影响到这些字段的内容。其余字

段会被文档分析模块处理后再被建立到索引中。

第 26 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

同时，这 5 个字段的内容在索引中都被引用到了同一个字段（\_entext\_ 字段）上，也就是说，平

台设立了一个名叫 \_extext\_ 的字段同时包含了这五个字段的所有内容。在默认查询中，这五个字段

都会被等价地被系统检索，即系统在用户不指定的时候，会返回在 \_entext\_ 字段的查询结果。如果

用户指定了只在某个字段进行查询，例如，只在论文标题中查询输入关键字的时候，系统会返回指

定字段的搜索结果。

。。。此处添加示意图

第 27 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

第六章 大规模数据的索引建立

在系统使用的完整的数据库中，论文的总量超过一亿两千万，因此索引建立的速度将会是索引

建立方式的主要考量因素。由于论文数据库的内容实际上在不断更新，随着时间的推移，数据库中

会添加最新的论文，现有论文的被引用量等信息也会改变，如果建立一次索引的时间过长，将会大

大降低该系统的实时性与实用性。考虑到如果要在 6 小时内建立完整的索引，每秒导入条目的数量

必须要大于 5500 条，这对建立索引的方法就有了很高的要求。经过多种方法的尝试后，本论文最终

用文件导入法代替了之前的数据库导入法，成功将索引条目导入的速度从最初的每秒 30 条左右增加

到了每秒 6000 条左右，成功做到了索引的实时建立与更新。

下面通过介绍并分析新旧两种导入方式，以解释针对这一问题的处理方法：

6.1 数据库导入法

数据库导入法是向 Solr 导入数据并建立索引的一种较为直观的方式，在 Solr 中的 DataIm-

portHandler 包中有该方式的相关接口。由于数据源是 MySQL 数据库，因此导入过程采用 JAVA 提供

的 MySQL 库中的 jdbc.driver 包进行驱动。

在本系统中，作为数据源的数据库结构为：

（图片待补）

因此，正如上一章中提到的，我们只要针对 DataImportHandler 包设计合适的 db-data-config.xml

文件，就可以实现数据库导入索引的操作。该文件的实现方式如下：

代码 6.1 从数据库中导入索引的 db-data-config.xml

<dataConfig>

<dataSource type=JdbcDataSource

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://\*.\*.\*.\*:\*\*/database

user=\*\*\*\*\*\*

password=\*\*\*\*\*\* />

<document>

<entity name=Papers query=select \* from Papers>

<field column=PaperID name=id />

<field column=OriginalPaperTitle name=OriginalPaperTitle />

<field column=OriginalVenueName name=OriginalVenueName />

<field column=PaperRank name=PaperRank />

<field column=PaperPublishYear name=PaperPublishYear />

<field column=ConferenceSeriesIDMappedToVenueName name=ConferenceSeriesID />

<field column=JournalIDMappedToVenueName name=JournalID />

<entity name=Information

query=select \* from PaperAuthorAffiliationswhere PaperID='${Papers.PaperID}'>

<field name=AuthorID column=AuthorID />

<entity name=Authors

query=select AuthorName from Authorswhere AuthorID = '${Information.AuthorID}'>

<field name=AuthorName column=AuthorName />

第 29 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

</entity>

</entity>

<entity name=Conference

query=select ShortName from ConferenceSerieswhere ConferenceSeriesID =

'${Papers.ConferenceSeriesIDMappedToVenueName}'>

<field name=ConferenceShortName column=ShortName />

</entity>

<entity name=Keywords

query=select \* from PaperKeywordswhere PaperID='${Papers.PaperID}'>

<field name=KeywordName column=KeywordName />

<field name=KeywordID column=FieldOfStudyIDMappedToKeyword />

<entity name=FieldsOfStudy

query=select FieldsOfStudyName from FieldsOfStudywhere FieldsOfStudyID =

'${Keywords.FieldOfStudyIDMappedToKeyword}'>

<field name=FieldsOfStudyName column=FieldsOfStudyName />

</entity>

</entity>

<entity name=PaperReferencesCount2

query=select PaperReferenceCount from PaperReferencesCount2where

PaperReferenceID = '${Papers.PaperID}'>

<field name=PaperReferenceCount column=PaperReferenceCount />

</entity>

</entity>

</document>

</dataConfig>

</requestHandler>

不幸的是，用这种导入方法从来都没有成功地导入过一次完整的数据库。因为数据导入速度过

慢，对于 1.2 亿条的数据，系统预计的导入时间超过了 30 天。而在服务器上将如此占用资源，尤其

是占用服务器数据库资源的进程连续运行 30 天是不可容忍的。由上一章可知，对于每一篇文章，需

要索引的域共有 14 个，共涉及到数据库的七张表。由于该方法的索引建立的过程是逐条建立，而不

是按表建立，因此需要通过表的连接（JOIN）方式将多张表的内容整合起来，再导入索引中。要整

合得到所需的全部 14 个域，其中作者 ID、会议简称、关键词名称、关键词 ID 需要进行一次表的连

接操作；作者姓名、研究领域名称需要进行两次表的连接操作。由于大部分数据库表的规模很大，进

行表的连接操作将会花费大量的时间。在实际操作中，使用该方法虽然可以完整建立功能完备索引，

但是索引的速度只有每秒 30 条左右，建立一次索引的时间将会超过 30 天。考虑到服务器资源的限

制与索引建立方式的可操作性，我们不得不抛弃这种方法，而寻找其它的解决方式。

6.2 文件导入法

在数据库导入法暴露了其致命缺陷之后，我尝试了很多改进方式，然而都没有从根本上解决这

一问题。究其原因，是数据库表本身连接运算的效率过低导致的。在理论上，计算笛卡尔积的连接

操作的复杂度达到了 O(M\*N)，经过 SQL 内部优化的 JOIN 操作复杂度也在 O(M+N) 之上。由于这

一部分涉及到大量硬盘读写，而硬盘的读写速度远远慢于内存与寄存器内的运算速度，这一典型的

I/O 密集型 (I/O bound) 工作的工作效率会被硬盘的读取速度大大限制。无论如何优化算法，或是给

以程序更多的内存，只要读入数据的方式不改变，运行的速度就不会有本质上的提高。

第 30 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

弄清了问题的本质以后，改进该部分的重点就到了如何减少硬盘的读写上。其中一个可行的方

式，也是论文中采用的方式，就是完全抛弃调用数据库操作处理数据，而将全部的数据操作都搬到

读入内存的哈希表中进行，我把这种方法叫做文件导入法。该方法的核心是先将数据库中的内容处

理为 XML 格式的标记语言文件，再用该文件作为数据源建立索引。由于处理得到的文件中每条论

文信息可以被逐条导入，因而这种方法完全避免了 I/O 密集的数据库操作，经过测试，使用该方法

的导入速度相比数据库导入法提升了 100-200 倍，使大规模索引快速建立这一目标得到了实现。

文件导入法的算法如下：

\*\*\*

在按此方法导入了数据之后，我们得到的 transed\_all.xml 文件便以逐条呈现的形式存储了所有

需要被索引的信息。此时我们以该 xml 文件为数据源进行索引创建的时候，就可以避免耗时的数据

库读取与链接操作，而可以通过顺序读文件的方法完成索引的简历。此时，db-data-config.xml 的配

置也会变得简单很多。注意，此时要以流形式 (stream = true) 导入数据，可以让系统逐行读取目标文

件，避免文件过大导致资源不足。

代码 6.2 从文件导入索引的 db-data-config.xml

<dataConfig>

<dataSource encoding=UTF-8 type=FileDataSource />

<document>

<entity

name=paper

processor=XPathEntityProcessor

forEach=/root/paper

url=/foo/bar/transed\_all.xml

stream=true

>

<field column=id xpath=/root/paper/PaperID />

<field column=OriginalPaperTitle xpath=/root/paper/OriginalPaperTitle />

<field column=OriginalVenueName xpath=/root/paper/OriginalVenueName />

<field column=PaperRank xpath=/root/paper/PaperRank />

<field column=PaperReferenceCount xpath=/root/paper/CitationCount />

<field column=PaperPublishYear xpath=/root/paper/PaperPublishYear />

<field column=ConferenceSeriesID xpath=/root/paper/ConferenceSeriesIDMappedToVenueName />

<field column=JournalID xpath=/root/paper/JournalIDMappedToVenueName />

<field column=ConferenceShortName xpath=/root/paper/ConferenceShortName />

<field column=AuthorID xpath=/root/paper/Author/AuthorID />

<field column=AuthorName xpath=/root/paper/Author/AuthorName />

<field column=AuthorSequenceOrder xpath=/root/paper/Author/AuthorSequenceOrder />

<field column=FieldsOfStudyName xpath=/root/paper/Keyword/FieldsOfStudyName />

<field column=KeywordID xpath=/root/paper/Keyword/KeywordID />

<field column=KeywordName xpath=/root/paper/Keyword/KeywordName />

</entity>

</document>

</dataConfig>

</requestHandler>

该导入配置文件比数据库导入法的配置文件要简单很多。这是因为此时的需要导入的内容都在

xml 树的同一个根节点下平铺展开，所以只要顺序读入该文件，以此读入 xml 树并获取对应域的内

第 31 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

容，就可以完成文档的导入工作。

这是一个典型的用空间换时间的算法，其本质类似于将硬盘中的内容映射到内存中，再利用内

存比硬盘高得多的读写速度完成任务。这种空间换时间以提高效率的方法在本平台中也有许多其它

的运用场景。例如在索引的构建中，如果将 60G 的索引文件映射到内存中，并直接在内存上执行索

引查询请求，会使查询的速度相比于将索引存放于文件中大大提高。但是在服务器中，内存空间是

一种比磁盘空间宝贵的多的资源，长时间占用 60G-120G（60G 为单机版索引总大小，120G 为分布

式索引总大小）的内存资源也是一项很大的花费，因此这种方法的使用需要经过慎重考虑。

第 32 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

第七章 搜索后台与前端的接口设计

在实现了搜索引擎的后台之后，接下来的步骤就是将其上线到网站上，并实现相应的用户交互

接口与界面。这一部分的功能在网站代码的 php 文件中实现。其实现流程图如下：

用户输入

提交输入

合法检查

输入合法？

向后台发出请求 成功响应?

错误页面

结果页面

是

否

否

是

图 7–1 网页处理流程图

Fig 7–1 Website Handler Procedure

当用户输入被确认合法（即去除特殊符号与停用词后不为空字符串）后，用户输入被封装到传

入 solr 服务器的 html 请求的 query 字段中，并访问 Solr 服务器以寻求其答复。发送给 Solr 的 HTTP

请求基本格式如下：

http://\*\*\*.\*\*\*.\*\*\*.\*\*\*:\*\*\*\*/solr/collection/select?indent=on&q=content&wt=json&page=2

&sort=PaperPublishYear desc&facet=false

事实上，我们还可以在 HTTP 请求中对 solr 服务器提出更多的要求。由于在本课题中没有得到

应用，故略去不讲。在上面那个例子中，在 HTTP 请求中声明的参数如下：

• q: 请求内容的主体，即用户搜索的内容。如果请求内容包含多个单词，用空格隔开的话，多

个关键词之间是或的关系，服务器会返回所有含有任何一个关键词的结果。如果在每个关键

第 33 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

词前面添加一个’+’ 符号，多个关键词之间是与的关系，服务器只会返回包含有全部关键词的

结果。另外，如果不指定搜索的范围，默认会在 \_entext\_ 域进行搜索（见第二章，平台配置）；

如果指定搜索的范围，会在指定的某个域中进行搜索。

• wt: 返回结果的格式。此处指定为 json，也可以根据需要改成 python, xml, php 等。这一部分不

影响返回的内容，但是会影响返回内容的组织格式，在网页对返回结果进行解析的时候，需

要选择与此处格式配套的解析策略。

• page: 翻页情况。被索引的论文总数超过一亿篇，这意味着一般搜索一个关键词都会看到超过

一万条的返回结果，而返回全部的搜索结果是不现实的。这时候就需要进行翻页。在默认情

况下，每页返回十条记录，通过指定该参数，可以让服务器返回结果中的第 10\*(page-1)+1 到

第 10\*page 条结果。

• sort：结果排序情况。默认条件下，我们设置的结果是按照论文的引用数排序，如果需要按其

他的字段，如按出版年份降序排序，就可以按上例所示修改排序方式。可以同时指定多个排

序方式，用逗号隔开，即当前一个指定项相同时，按后一个指定项的顺序进一步排列结果。

• facet：是否统计结果。Solr 服务器提供了统计结果的功能，本搜索平台也在网页上对结果统

计进行了相关的界面实现。但是统计结果功能会在很大程度上影响响应速度，所以当对返回

结果速度要求较高的时候，可以将此参数设为否，即关闭统计功能。

如果 Solr 服务器正确响应，对于以上请求将会以 Http Response 的形式返回一个.json 格式的文

件。该文件中即包含了网页需要显示的所有信息。该文件的结构如下：

代码 7.1 返回文件格式

\{

responseHeader:\{

status:0,

QTime:1442,

params:\{

q:content,

indent:on,

page:2,

wt:json,

facet:false\}\},

response:\{numFound:332899,start:0,docs:[

\{

PaperReferenceCount:0,

ConferenceShortName:,

\_entext\_:[,

,

contents contents,

Contents-May 2015,

2015],

AuthorID:[7AABEB29],

OriginalVenueName:[],

AuthorName:[contents contents],

ConferenceSeriesID:,

OriginalPaperTitle:[Contents-May 2015],

JournalID:,

AuthorSequenceOrder:[1],

第 34 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

PaperPublishYear:2015,

id:5F993E0C,

\_version\_:1554052763125547020\},

......],

highlighting:\{

5F993E0C:\{

OriginalPaperTitle:[<font color=#196600>Contents</font>-May 2015]\},

......

\}\}\}

在该响应文件中，responseheader 里显示了我们查询的参数，response 中显示了响应的内容。其

中，response 中的 docs 列表中将会返回在该次查询中返回的十条结果，我们只要解析该 json 文件，

并把对应字段的内容加上对应字体的 html 标签在返回页面显示，就实现了搜索结果的展示功能。除

了基本内容之外，网页还有结果高亮和结果统计两个扩展功能，进一步扩展了网页的显示效果。

7.1 结果高亮

结果高亮的目的是用不同的颜色或字体表现用户搜索的关键词部分，使之更加明显。在系统构

建的过程中，关键词高亮的功能最初是通过正则表达式匹配替换的方法实现的，即在返回结果中正

则匹配出用户输入关键词的部分，然后用两边加入不同颜色的 HTML 标签的方式实现。然而正则表

达式匹配法在高亮上存在一个两难抉择的问题：

• 如果采用宽松的匹配方式，即只匹配搜索关键词而不管关键词两边的内容，可能会出现错误

匹配的情况。例如，用户搜索单词 get 的时候，和关键词无关但是包含了该单词的单词会被全

部高亮，例如 together, forget 等，但是因为本搜索平台是以词为单位进行检索，因而这些结果

并不是因为被高亮的部分而被查询到的，这样一来，这种高亮显示就会对用户产生误导，也

偏离了关键词高亮这一功能的设计初衷。

• 如果采取严格的匹配方式，即匹配搜索关键词，而且要求关键词两端是字符串边界，空白字

符或符号的话，可以避免上述的问题，但是名词或动词变形的情况就无法被匹配到。例如动

词结尾 +s, +es, +ed，或以 y 结尾的动词的过去式变形等。这样一来高亮的功能就会受到一些

限制，同样会影响到高亮功能的表现。

考虑到以上两个问题用正则匹配法难以同时解决，搜索平台不得不采取另一种方法解决关键词

高亮的问题。平台采用的方法为词根匹配并记录位置的方法，由于在搜索引擎中，系统对用户请求和

文档内容本身都进行了词根化的操作，而词根也是搜索引擎进行检索的最小单位，因此我们只要记

录词根在文档中出现的位置，再于词根化之前的文本中的相同位置进行高亮标记，即可实现同时解

决了上述两个问题的高亮算法。在上面的示例文件中可见，虽然用户的查询内容是content，但是首字

母大写的复数形式 Contents 也被正确高亮，同时，由于包含了 content 的其它单词，例如 subcontent，

和 content 拥有不同的词根，因此不会被高亮标记。

在 json 返回结果中，我们将 highlighting 域的对应内容代替 docs 域的对应内容放在网页上进行

展示，即可达到关键词被高亮显示的效果。

第 35 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

7.2 结果统计

结果统计功能的目的是统计搜索结果的分布情况。本论文第二章中已经提到过，配置文件中对

搜索平台搜索结果的默认统计域有五个——出版年份，关键词，作者，会议名称，期刊名称。其中

对于每篇文章，会议名称和期刊名称不会同时出现。统计功能即是统计搜索结果中出版于各个年份

的论文数目是多少，拥有某关键词的论文数目是多少，还有不同作者与搜索关键词相关的论文数量

数目等等。结果统计功能可以让用户不需要浏览所有的搜索结果，就可以对搜索结果的特点产生直

观的印象。

如果在 html 请求中将 facet=false 改为 facet=true，即可打开结果统计功能。此时，服务器返回的

json 文件中会增加一个 facet 键对应的内容：

代码 7.2 结果统计

\{

facet\_counts:\{

facet\_queries:\{\},

facet\_fields:\{

PaperPublishYear:[

2013,23649,

2014,22127,

2012,20761,

.....],

KeywordID:[

017C8A77,9940,

08EE83EF,5945,

09AEBB9C,4648,

200524E7,4165,

.....],

.....\},

facet\_ranges:\{\},

facet\_intervals:\{\},

facet\_heatmaps:\{\}\}

\}

如上例所示，由于统计功能的相关参数都在默认配置文件中指定，因此 facet\_queries 等域的内

容均为空。同时，facet\_fields 的内容显示了与该关键词相关的论文的分布情况，例如出版于 2013 年

的论文的数量为 23649 篇，2014 年的数量为 22127 篇等。

得到了该统计结果，接下来就是要在网页上对其进行可视化显示。在平台中，我们使用了

ACharts.js 作为绘图工具绘制统计结果图表。考虑到在这五个被统计的域中，对于年份人们关心的

是数量随时间变化的信息，而对于其他四个域人们关心的是数量排名信息，因此我们对年份域的结

果按年份排序，对其他域的统计结果按统计数目从多到少排序，再用折线图与柱状图进行分别呈现。

由于结果统计功能复杂度较高，花费时间多，因此开启了统计结果的网页响应速度会比关闭该

功能的响应速度慢很多。为了解决这一问题，平台采用了 Ajax 异步加载的方法，让搜索结果和统计

结果异步完成呈现。在用户提交查询请求后，平台会先向搜索服务器发送一次关闭统计功能的请求，

先快速显示查询结果，此时统计结果的图表显示为加载状态。查询结果显示完毕后，平台再向服务

器发送一次打开结果统计的请求，得到响应后再绘制结果统计的各项图表。

第 36 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

第八章 查询系统的分布式部署

完成了前面几章叙述的内容后，搜索平台经过测试后就已经可以正式上线了。然而在系统实际

运行的过程中，却发现系统有一些因为只有单台服务器而产生的问题。例如：

1. 当服务器负荷较大，如正在运行其它高系统占用率服务时，平台的搜索速度会有极大地减缓。

2. 当查询平台的进程因内存溢出等原因崩溃或服务器计算机重启时，网站的搜索结果会直接报

错。

这些实际问题的产生也体现了搜索平台对分布式服务器的需求。对于大规模的查询系统，单台

服务器往往有它显而易见的局限性，例如负载上限较低，稳定性较差，容错性低等。使用多台服务

器部署分布式云架构搜索平台，是解决单台服务器局限性的一种很好的方式。这种架构有如下优点：

1. 实现了多台服务器查询时的负载均衡，使搜索服务器可以动态分配计算资源，达到性能最大

化；

2. 实现了多台服务器上配置文件的集中管理，将多台服务器的配置文件统一挂在在一个节点上，

避免了多份配置文件管理上的不便；

3. 实现了多台服务器运行状态的统一监管，在任意节点上可以查看所有节点的运行状态；

4. 采用了分布式的自组织架构，并没有严格意义上的主服务器，任何一个服务器挂掉，都不会

影响到整个系统的正常运行；

5. 实现了服务进程的自动恢复，当任何一台电脑需要重启或关闭服务进程时，直接关闭该进程

不会中断搜索服务的提供，并且在服务进程重新启动后，进程能自动恢复到云上；

6. 实现了索引文件的备份。对于索引文件的每一个分片，其备份数目都与云平台上节点数目相

同，这在保证了系统容忍任意 (N-1) 个节点同时故障的同时，也对索引文件实现了多次备份，

避免了某几个服务器硬盘损坏导致的数据不可恢复地丢失。

当然，要实现拥有这么多功能的云架构搜索平台，就不可避免地需要大大增加整个系统的复杂

度。虽然目前互联网上有一些相关的资料，但是大部分资料都是让分布式服务运行不报错就浅尝辄

止，而几乎没有资料讲述了完整地实现一个功能完整的分布式搜索服务器的方法。因此在做这一部

分工作的时间里，我付出了大量的时间去摸索实现分布式服务的各个环节，也花费了比预期更长的

时间才完成了分布式搜索平台的所有功能。

实现分布式搜索平台的第一步就是要设计系统的架构。架构设计的目的是要在尽量不改动之前

实现的功能模块的基础上为系统引入分布式的服务，同时要使分布式的系统尽可能地稳定，功能强

大，并且易于维护。

8.1 分布式系统的架构

要想实现搜索平台的分布式部署，就必须引入分布式程序的协调工具。在本平台中，我们使用

Apache Zookeeper 来进行分布式服务的协调。Zookeeper 在本应用中，显式地可以完成配置文件云节

点的挂载以实现配置文件的统一管理，隐式地可以完成主服务器选举、负载均衡、同步数据、锁死

避免等分布式服务所需要的功能。经过初步设计后，分布式平台的架构设计如下：

第 37 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

图 8–1 云服务器初始架构

Fig. 8–1 Solrcloud Simplified

在这里，我们把原来单一的 Solr 实例 (Instance，又叫 Core) 分为四个节点，并由一个 Zookeeper

服务器进行统一管理。同时，我们原本的单一索引文件被分成了两个片 (Shard)，每一个片分为两个

相同的备份。这样一来，每个 Solr 节点里恰保存一份索引，并负责与该份索引相关的索引工作。我

们将 Core1 与 Core3 放在同一台服务器上，Core2 个 Core4 放在另一台服务器上，四台服务器上的两

片索引文件和两片备份索引文件共同组成了索引集合 (Collection)。

由于在实例中保存的配置文件难以同步修改，因此在这四个 Solr 实例中均不保存任何配置文件，

所有的配置文件我们统一保存在 Zookeeper 的数据节点中。在索引构建的阶段，四个实例分别读取

Zookeeper 数据节点中的配置文件，确定自己的索引结构，并由 Zookeeper 内部算法决定将每一篇文

档转发到哪个分片中进行索引；在查询阶段，Zookeeper 决定将查询请求分配给哪些较为空闲的节

点，并通过配置文件决定给 Solr 实例的查询请求格式，由选举出的主实例 (Leader) 进行结果整合与

发布。

当四个节点中的任意一个节点因为意外（例如索引文件损坏，进程意外终止等）无法正常工作

时，Zookeeper 会自动检测到节点的异常，并将其标记为宕机（Down）状态或离线（Gone）状态，此

时整个系统的服务并不会终止，而是会由剩下的节点分配承担所有任务。由于目前架构中任意一台

服务器上都保存着一份所有索引的备份，所以只要不是所有的计算机服务器全部宕机，完整的服务

功能都会保持。当意外关闭的节点被重新注册回 Zookeeper 时，Zookeeper 会尝试恢复该节点并使其

回复正常工作。

以上的架构看似很完美，然而仍然存在一个严重的缺陷——那就是 Zookeeper 节点的单一性会

导致其反而成为系统鲁棒性的短板。换句话说，虽然对于目前的架构任意 Solr 节点的宕机都不会造

成服务故障，但是一旦 Zookeeper 服务本身出现故障，所有的服务都会跟着瘫痪。解决这一问题的方

第 38 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

法有两个，一是将 Zookeeper 服务运行在几乎不需要维护重启的专门服务器上，二是针对 Zookeeper

服务器本身建立一个分布式架构。用通俗的话说，第二种解决问题的方法，就是用 Zookeeper Cloud

来管理 Solr Cloud。该改进的架构如下图所示：此时，Zookeeper 不仅管理着四个 Solr 节点，同时也

图 8–2 云服务器改进架构

Fig. 8–2 Solrcloud Advanced

管理着两个 Zookeeper 节点。与 Solr 节点自动选取 Leader 类似，Zookeeper 节点的 Leader 也是由内

部自动选举得到的。两个 Zookeeper 节点的存在分担了 Zookeeper 服务器宕机的风险，使得其中任何

一个 Zookeeper 服务器故障都不会影响到系统整体的正常工作。一个显而易见的好处是，在这个架

构下我们可以只配置两台服务器计算机，并在每台计算机上部署一个 Zookeeper 节点和两个 Solr 节

点，此时任何一台计算机的重启都不会终止服务的正常提供。而如果不配置两个 Zookeeper 节点，运

行着 Zookeeper 服务的计算机是不能重启的，否则整个服务就会崩溃。

但是这个方案和第一套方案相比也不是十全十美的。第一套解决方案能保证系统在上线后，能

够用一套十分简单的流程来维护系统的所有功能，而对于第二套解决方案架构的系统，其维护过程

会比第一套解决方案复杂很多，在非专业人士操作维护时可能会让服务器产生更多的故障。因此这

里只是在理论上说明了第二套流程的可行性及其优势，在实际搜索平台的实现中，我们并没有采用

云架构部署 Zookeeper 本身。

8.2 分布式系统的部署

这一部分介绍云服务器初始架构的部署方式。具体的部署命令可以参见附录 1 中，上面有详细

的罗列。

第 39 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

首先，我们需要配置并启动一个 Zookeeper 服务。Zookeeper 服务的配置需要指定 5 个参数，我

们通过修改 zoo.cfg 文件以完成此处的配置。

1. tickTime：一次同步的时间周期，此处设置为 2000ms；

2. initLimit：第一次同步过程花费时间周期数的限制，此处设置为 10；

3. syncLimit：一次同步过程的时间周期数限制，即发送请求与得到相应之间的最大周期数间隔，

此处设置为 5；

4. dataDir：存放快照数据的目录，根据需要设置；

5. clientPort：服务运行的端口，所有的 Solr 节点都要通过这个端口注册到 Zookeeper。

配置好这些参数后，我们就可以启动 Zookeeper 服务了。启动 Zookeeper 服务后，我们可以打开

其客户端（Client）查看内部文件情况，由于是新挂载的节点，所以里面是没有任何文件的。针对之

后需要挂载的 Solr 服务，需要在 Zookeeper 上挂载一个空节点，这里起名为 acemap 节点。

接下来要将 Solr 服务器注册到 Zookeeper 上。我们在两台服务器上各启动一个 Solr 服务，然后

在每一个 Solr 服务上运行两个实例节点。在这一过程中，首先需要配置 Solr 服务的 host 地址，由于

本平台中两台服务器并不在同一个内网上，所以我们使用公网地址作为服务的 Solr 地址。然后，我

们需要指定服务以云模式启动并挂在到 Zookeeper 的 acemap 节点上，并指定为其 Java 虚拟机分配的

内存。

第一台服务器上的 Solr 服务启动后，Zookeeper 的 acemap 节点上会自动保存一些基本的配置文

件，打开其中的 live\_nodes 文件夹，可以看到对应与启动服务 host 名称的记录文件。启动第二台服

务器上的 Solr 服务后，live\_nodes 文件夹里会显示两个服务各自的记录文件。

之后，我们需要在两个 Solr 服务组成的集群上创建一个新的索引集合（Collection），并以 2 的冗

余度创建两个分片。此时，这四个分片会被自动分配到四个 Solr 节点中，同时每台服务器上的 Solr

服务都会自动分配到两个不同的分片。

最后，我们需要实现配置文件的统一管理功能。这一功能可以借助 zkcli.sh 进行实现。我们先将

所有的配置文件统一放入文件夹中，用 zkcli.sh 的 upconfig 方法可以将本地的文件夹传入 Zookeeper

对应节点上，downconfig 方法可以将 Zookeeper 节点上的文件下载到本地。在文件夹被挂载到节点上

之后，我们可以用 linkconfig 函数将节点与 Solr 云中的 collection 对应起来。

至此，Solr 服务器的分布式部署已经基本完成，我们可以通过访问任何一个 Solr 服务器来管理

这个分布式的集合（collection）。

8.3 分布式系统的配置

分布式系统的配置方法和单台服务器的配置方法大同小异。在集合整体的视角下，四个子节点

组成的整体可以视作一个大的单台服务器，因此，系统配置的过程和单台服务器一样，需要先配置

solrconfig.xml 文件，再进行索引结构设计与文档导入，最后进行查询指令的设计。不同的是，在配

置文件与索引结构设计完成后，注意需要用 reload 指令确保指令被同步到了每个节点之中。

此外，由于索引被分成了两个独立的片，因此查询指令中新增了一条&shard=XXX的命令，用以

只在某个片中搜索结果。更进一步地，可以指定只在某个shard的某个特定的备份中进行查询。这种查

询方式称作分布式查询。 （参考资料https://cwiki.apache.org/confluence/display/solr/Distributed+Requests）

第 40 页共 53 页

学术搜索引擎大规模查询系统的建立与优化

既然 Solr 云架构拥有分布式查询的功能，那就说明 shard 的构建过程除了交给系统自动进行外，

也可以人为地进一步自定义。例如，在集合创建时，可以用 router.name 属性在决定每一条文档被分

发到哪一个具体的片上。例如对于本学术网站的搜索平台，router.name 可以由论文属于计算机领域

或是非计算机领域决定，通过将计算机领域的论文索引到其中一个片，非计算机领域的论文索引到

另一个片，就可以用分布式查询的方法高效的在计算机或非计算机论文这两个子集中分别查询论文。

但是由于实际上非计算机领域和计算机领域论文这一划分方式并不均匀，前者的数量实际上会远远

多于后者，所以会导致资源分配不均匀的问题。我们可以通过添加服务器，并利用片分割（Shard

Splitting）的方法将较大的片进一步分割存储到更多的服务器中，以达到负载均衡