浙江大学

本科生毕业论文（设计）
开题报告

学生姓名：晁建嵩

学生学号：3062121037

指导老师：耿卫东

年级与专业：06 数字媒体

所在学院：计算机学院

一、题目：Bingo!——基于 VisualRank 模型与相关文本的商品图像搜索引擎

二、指导教师对开题报告、外文翻译和文献综述的具体要求：

对于开题报告：要求给出合理的技术路线和清晰的工作计划，并进行文献的收集分析；对于文献翻译和文献综述：符合学校要求，字数 3000 字以上。

指导教师（签名） 耿卫东

2010 年 03 月 20 日

毕业论文（设计）开题报告、外文翻译和文献综述考核

答辩小组对开题报告、外文翻译和文献综述评语及成绩评定：

成绩比例	开题报告 占（20%）	外文翻译 占（10%）	文献综述 占（10%）
分 值			

开题报告答辩小组负责人（签名）

年　　月　　日

目 录

本科毕业论文（设计）开题报告

1. 课题背景

在前 Google 时代，尤其是 Google 出现的前几年，Web 信息量急剧增长，Web 搜索引擎的发展遇到了瓶颈，当时的主流搜索引擎主要有两类，人工索引搜索引擎和全自动搜索引擎。人工索引搜索引擎可以提供十分准确和高质量的信息，但是维护成本很高，更新速度慢而且远远不能覆盖所有领域；自动搜索引擎基于查询词的匹配。然而随着 Web 页面的爆炸式增长和作弊页面的逐渐增多，这种搜索引擎已经很难查询到真正高质量的搜索结果。就在此时，来自 Stanford 的两位天才学生，Sergey Brin 和 Lawrence Page 提出了一种著名的网页排名计算模型，并以此为基础创建了 Google，从而掀起了搜索引擎领域新的革命，把互联网带入了新的时代，这个神奇的模型就是 PageRank。PageRank 的本质想法十分简单，Sergey Brin 和 Lawrence Page 基于对互联网具有洞察力的观察，提出了以下的假设：页面 A 含有指向另一个页面 B 的链接，则视为页面 A 对页面 B 投了一票。同时他们还考虑到了不同网页的权威性以及网页自身的链接总数等诸多因素，把整个 Web 构建成了一幅索引图，在此之上进行矩阵运算，从而得出每张页面的 PageRank 分值，这个分值在最终搜索结果的排序过程中起到了相当大的作用，它使得质量高的网站排名能够排在靠前的位置，有效地解决了之前的搜索引擎遇到的难题。网页搜索出现不久，出现了很多种垂直搜索引擎，像音乐搜索，图片搜索等。图片搜索虽然搜索结果是图片，但是直到今天，主流的图片搜索引擎仍然完全依靠相关文本来对图片相关性进行排序，这主要有三点原因：1 基于文本的图片搜索技术已经十分成熟，并且在实际中取得了很大的成功。2 对图片进行类似人眼功能的识别到现在还是极其困难的一个领域。一些特有领域的识别，像人脸识别的研究已经得到了一些成果，但是对通用图片集而言，还有很长的路要走。3 对图片的处理速度要远远低于对文本的处理速度，这种计算量对于大规模的搜索引擎而言，是难以承受的。近期，Google 发表了一篇论文，提出了一种新奇的想法：将 PageRank 模型应用于图片搜索引擎。其主要思想是如果图片 A 与图片 B 相似，则可以认为图片 A 对图片 B 投了一票，将图片的相似性作为类似计算 PageRank 时所用的引用图的边值，从而计算每张图片的 VisualRank，这是完全离线的过程，为其在实际中的运用提供了可能性。

2. 目标和任务

（首先，我把系统名称命名为 Bingo！这有两层含义：1 Bingo 本身的意思很像搜索引擎搜到人们想要的结果时的感觉。2 Bingo 可以被分解为 Bing 和 Google 的前缀，系统本身也从 Google 和 Bing 得到了许多重要的数据。）

Bingo！将是一款基于 VisualRank 模型（将 PageRank 应用于图片搜索的新模型）与传统相关文本的新型商品图片搜索引擎。然而 VisualRank 是一种开放的模型，有很多方面（例如比较图片相似）需要去探究，而构建整个搜索引擎更是需要很多其他方面的研究。我不期望短期之内 Bingo！能够将 VisualRank 模型应用到

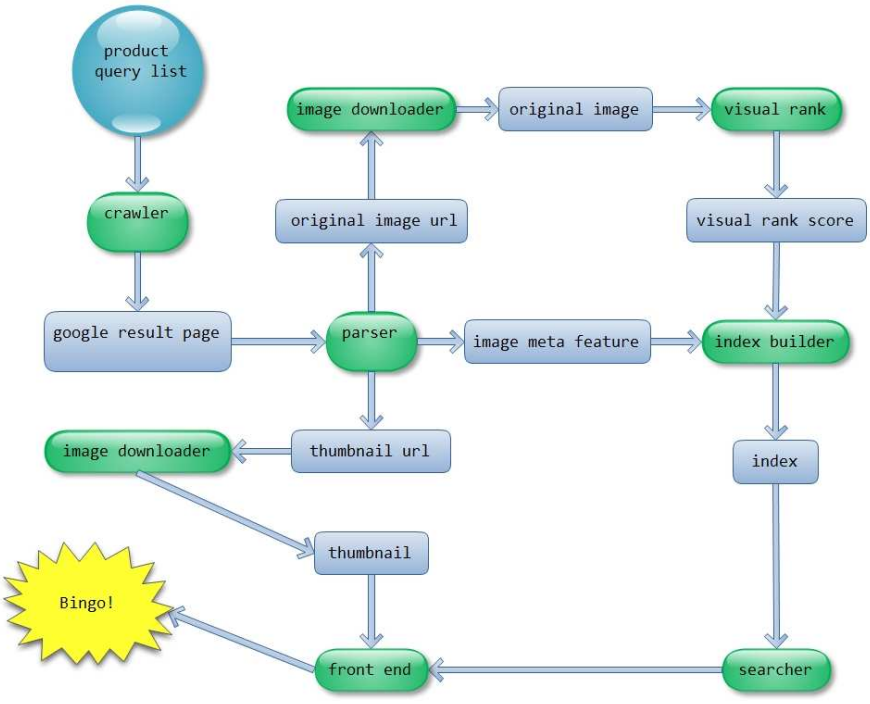
大规模的搜索引擎中，所以我将它限定在一个特定的领域内，就是商品图片搜索引擎。Bingo!会收集时下最流行的商品查询并将已有的图片搜索引擎（像 Google 或 Bing）的基于文本的搜索结果进行抓取，构建自己的图片数据库，缩略图数据库和图片属性数据库，并对图片进行相似度分析构建成类似于 PageRank 的引用图的相似图，在此之上进行 VisualRank 的计算，然后与图片的属性数据合并构建索引，再通过查询系统衔接索引数据和前端，组成一款完整的商品图片搜索引擎，达到比时下主流的基于文本的大规模图片搜索引擎更好的搜索结果。

3. 可行性分析

传统的基于文本的搜索引擎已经有很多成功的先例，然而，编写搜索引擎仍然是充满挑战的工作，通过限定引擎的规模与领域，以及利用已有的搜索引擎的计算结果作为数据源等方式，引擎中的一些组件的开发得到了不小的简化，也让我有更多精力来进行 Bingo！的核心技术——VisualRank 的开发。

另一个充满挑战的领域是图片的相似度判断，这个领域有许多发表的论文，我可能需要将比较大的精力放在如何充分的利用这个精美的模型，因为这才是改进 Bingo!搜索结果质量的最关键因素。综合整个系统的开发难度和理论深度，个人认为三个月的时间应该足够构建小规模的商品搜索引擎。

4. 研究方案和关键技术考虑



上图为 Bingo! 的总体架构，绿色表示 Bingo! 的组成模块，蓝色为 Bingo 需要处理或存储的数据

我将对其中的关键技术进行一一阐述：

1 crawler：爬虫对搜索引擎来说是难度最大的组件之一，因为需要处理很多人们通常不会想到的异常情况并且需要和不同的 Web 服务器和域名服务器打交道。然而因为 Bingo! 限定了搜索的领域与规模，并且原始网页数据来自于 google 产生的动态页面，所以设计得到了不小的简化，但是需要注意的是，google 会对抓取其搜索结果页的程序进行识别并禁止此类抓取，所以这里需要一些技巧来骗过 google 的判断，Bingo! 选用 curl 作为下载页面的工具，同时设置 user-agent 为 Web 浏览器，这样就可以绕过 Google 的防盗系统下载到 google 的搜索结果页，这样既保证了来源数据的规范和可靠，又降低了系统的开发难度。

2 parser：与 crawler 一样，由于限定了爬取网页的来源，parser 的设计难度也得到了有效地降低，Bingo! 针对 google 的结果页进行 parser 程序，google 的搜索结果页含有十分丰富的信息，解析这些数据需要对页面的结构有深入的了解，并对可能发生的异常情况进行处理。解析出的内容主要有三部分：原图 url，缩略图 url 和图片属性，原图的 url 会送给 image downloader 进行下载和存储，随后用来进行 VisualRank 计算，缩略图的 url 也会被送给 image downloader 进行下载和存储，之后用来为前端提供缩略图，图片的属性信息会送给 index builder 和 VisualRank 一起构建索引。

3 visual rank：这是 Bingo! 的核心技术所在，也是 Bingo! 区别于普通图片搜索引擎的本质所在。

visual rank 有两个独立的子模块组成：图片相似度计算模块和 PageRank 矩阵计算模块。计算图片相似度是一个相当具有难度的问题，具体的算法还没有确定，但是从架构上来说，图片相似度判断的模块会被设置成类似于插件形式，这样可以为 Bingo! 实现几种目前证明有效的方法，并根据最后的搜索结果进行替换与改进。对于 PageRank 计算模块，首先会利用图像相似度模块计算的结果构建 PageRak 需要用到的矩阵，矩阵的 Mat[i][j]即表示图片 i 与图片 j 的相似度，然后对矩阵进行转置并按列进行归一化，之后计算矩阵的关于特征值 1 的特征向量（这里可能要考虑一个阻尼参数，可能构造的矩阵会略有不同）。求这个特征向量可以将问题转化为线性方程组求根，然后利用高斯消元法求解，这种算法精度高但是计算量比较大，如果系统在运行的过程中，这一步的计算成为了效率的瓶颈，那么此处会考虑采用数值方法来近似计算 VisualRank。最后的结果是会对每一张图像计算出一个静态分，这个分数将被存储，并为之后构建索引做准备。

4 index builder：之前 Bingo! 进行了许多相关的计算步骤，得到了许多计算结果，这些方方面面的数据将在 index builder 中合并然后构建出索引数据，索引数据中会纪录所有的商品查询词，并将这些查询词放入内存中，每个词在放入内存时记录了指向相应图片集合的指针，每张图片的 visualrank 以及相关属性，索引的数据结构需要精心设计以保证 searcher 能够高效的利用索引进行查询。

5 searcher：在此之前，我们所涉及的所有模块对搜索服务来说都是离线的过程，而 searcher 是一个将离线处理的索引数据与前端的查询进行衔接的模块，searcher 需要负责接受前端的搜索请求，高效的利用索引数据查询结果并对搜索结果进行排序然后返回搜索结果给前端，这里需要注意的是前端的查询可能会并发执行，所以 searcher 必须对这种情况进行有效地处理，并考虑很多异常情况以增强系统的鲁棒性，因为 searcher 是在线的模块，所以系统的鲁棒性直接影响 Bingo! 的可用性和用户体验。

6 front end：前端负责将用户的查询传给 searcher 并将 searcher 返回的结果在页面上得以显示，页面的设计不是本人的强项，所以 Bingo！的主页只需要简明易用即可，暂时不做过多的要求。

5. 预期研究结果

Bingo！的设计目的就是能在自己索引的数据范围内提供比现有的基于文本的主流图片搜索引擎更好的搜索结果。由于 PageRank 模型在网页搜索中大放异彩，我相信 VisualRank 也能够在图片搜索中起到关键的作用，使得搜索质量高于现有的基于文本的图片搜索引擎。

由于对图片搜索引擎的评价不是一件很简单的事，我会尽量多选取一些数据集并邀请尽可能多的用户反馈用户体验，希望结果和我的预期一样。

6. 进度计划

2010.03.01—2010.03.05 确定毕设题目

2010.03.06—2010.03.16 翻译相关领域论文

2010.03.17—2010.03.21 总体架构设计，书写开题报告

2010.03.22—2010.03.31 完成 crawler 模块编码，启动 crawler

2010.04.01—2010.04.07 完成 parser 模块编码，在爬取下的页面集上进行操作，得到图片 url 列表，缩略图 url 列表和图片属性数据

2010.04.08—2010.04.10 完成 image downloader 模块编码，开始 download 原图数据和缩略图数据

2010.04.11—2010.04.20 完成 visual rank 模块编码并对图片数据集计算 VisualRank

2010.04.21—2010.04.30 完成 index builder 模块编码，构建索引

2010.05.01—2010.05.10 完成 searcher 模块编码，实现无图形界面搜索

2010.05.11—2010.05.20 完成前端模块，联调，实现全部系统功能

2010.05.21—2010.05.31 对系统进行最后测试并总结各部分工作完成毕业论文

本科毕业论文（设计）文献综述

随着 Google 的诞生，有关搜索引擎及其相关领域的研究取得了很多进展. Google 的创始人 Lawrence Page 和 Sergey Brin 在 Google 诞生时写了一篇文章: "The Anatomy of a Large-Scale Hypertextual Web Search Engine"，两人在这篇文章中对大规模搜索引擎的方方面面进行了详细了分析和论述,并提出了新型的网页排序模型 PageRank,

随后的一篇论文"The PageRank Citation Ranking: Bring Order to the Web"对这种模型进行了详细的描述. PageRank 巧妙地利用了 Web 的超链接结构. 这种模型基于这样的假设：如果页面 A 中含有指向页面 B 的链接，则说明页面 A 认为页面 B 的内容对它有意义，可以视为页面 A 为页面 B 投了一票，这样将页面投票的结果统计出来后，每张页面会得到一个投票数，作为它的重要性判定分值，这个分值就是 PageRank. 具体的 PageRank 定义如下:
设 PR(A)为页面 A 的 PageRank，C(A)为页面 A 中含有的链接总数，d 为一个阻尼参数，则：

PR(A) = (1-d) + d(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))

这个公式非常容易理解，但是蕴含了很多对 Web 具有洞察力的观察. 首先，对于每个指向页面 A 的页面的 PageRank 需要取除以此页面的链接个数，因为链接个数越多，用户从此页面链接至此页面的概率越低，而 PageRank 的内在含义本身就是用户通过超链接访问这个页面的概率. 第二，如果一个页面本身的 PageRank 很高，则其指向的页面会获得很高的加分，而不是把所有的反向链接视为相同权重. 比如，一个被 Google 主页引用的页面必然是重要的页面，其价值肯定比普通小网站的大。第三，公式中有一个阻尼参数 d，这个参数的内在含义是当用户浏览互联网时，比如正在浏览页面 A，那么很可能不会通过页面 A 上的任何超链接来继续访问下一个页面，这个概率就是阻尼因数所表达的意义。 然而，阻尼因数的作用不仅仅限于此，在论文"The pageRank Citation Ranking: Bring Order to the Web"中解释了这个参数的数学作用。 计算 PageRank 相当于求 Web 引用图构造的矩阵的的主特征向量，但是图中会有一种叫做 Rank Sink 的现象，例如页面 A 和 B 都只含有指向对方的链接，同时页面 C 指向 A，则 A 和 B 只会吸收 PageRank 而不会贡献出 PageRank，这就会导致 C 的 PageRank 为 0，不论它是多么重要的页面，所以需要把 Web 引用图构造为强联通分量，这样简化了 PageRank 的计算。 PageRank 可以通过纯数学方法来求得，但是 Web 页面如此之多，这种计算方法的效率不能满足要求，所以 Google 采用了迭代的工程算法来近似求解。另外，因为 PageRank 的语义是用户访问 Web 某页面的概率，所以计算时令所有页面的 PageRank 和为 1. 有关 PageRank 的计算有很多相关的论文，Stanford 在他的"Effcient Computation of PageRank"中对高效的进行 PageRank 的进算进行了详细的阐述.

信息检索系统的相关工作在很多年前就已经广泛开展。然而，大多数信息检索系统建立在小规模的同质集合上，例如科学论文或者特定主题的新闻。实际上，信息检索的主要基准，文本检索会议，基于相当小的易控集合。所谓的大规模的标准也只有 20G，而 Google 爬取的 2 千 4 百万网页具有 147GB。在 TREC 上效果良好的方法在 Web 上不会产生很好的效果。例如，标准的向量空间模型会根据给定查询和词频向量尝试返回与查询最接近的文档。在 Web 上，这种策略经常会返回非常短的文档，只含有查询词加上一些其他内容。比如，我们看到过搜索引擎对“Bill

Clinton”这个查询返回的结果只含有“Bill Cliton Sucks”和图片。一些人主张在 Web 上用户应该更清楚的指明想要的内容并且在查询中加入更多的词。Google 强烈反对这种观点。如果用户查询像“Bill Clinton”这样的词，他们理应得到合理的结果，因为关于这个话题有大量的高质量信息。由这些例子可知，标准的信息检索方法为了能够有效的应用于 Web 需要进行扩展。

Web 是一个庞大的无组织异质文档集合。文档间的内在差异很大，能够得到的外部元信息也各有不同。例如，文档本身的语言（人类语言或程序），词汇（email 地址，链接，邮政编码，电话号码，产品编号），类型或格式（text，HTML，PDF，图像，音频）都会存在内在的差异，甚至可能是机器产生的（日志文件或数据库的输出）。另一方面，我们把外部的元信息定义为一种可以推测文档内容的信息，但是不是文档的一部分。外部信息会含有来源的权威性，更新频率，质量，受欢迎程度和引用。不仅外部信息源的信息可能不同，需要考虑的事情的数量级也不同。例如，把一个像 Yahoo 这样的每天接受上千万次访问的主流主页和一篇十年才有人看一次的生涩的历史性文章相比。很明显，这两种信息在搜索引擎中该区分对待。

另一个 Web 与传统的有组织集合重大的不同是 Web 无法控制人们放在上面的内容。这种自由性对搜索引擎有很大的影响，搜索引擎必须处理一些为了盈利故意误导搜索引擎的公司的行为，这是一个严重的问题。这个问题传统的封闭式信息检索系统是无法解决的。甚至有很多公司为了盈利专门研究怎样操纵搜索引擎。

网页搜索出现不久后，诞生了很多种垂直搜索，其中就有图像搜索，很多年来，成熟的图像搜索引擎都是基于文本的，Google 的 Shumeet Baluja 在论文"PageRank for Product Image Search"中说明了其中的原因：1 基于文本的图像搜索有很多相关的研究和实践，并且已经取得了巨大的成功. 2 图像处理和分析仍然没有取得突破的进展. 3 图像的处理速度比文本来说慢的多，大规模的处理很难应用到实践中去. 但是这篇论文提出了一种非常新奇的想法，就是把一种和 PageRank 相似的思想带入图像搜索领域，称之为 VisualRank. VisualRank 基于这种思想：如果对某一个 query，图片 A 和图片 B 很相似，则浏览图片 A 的人也很有可能希望浏览图片 B，所以相当于图片 A 给图片 B 投了一票. 之后的过程和 PageRank 就十分相似了. 作者基于这种想法，对 Google 的某些 query 结果进行重新排序，并得到的相当好的结果，因为计算 VisualRank 是离线的过程，所以很好的掩盖了计算速度慢的缺点，使其在实际中的运用成为可能. 判定两张图片的相似涉及很多工作。

Shumeet 在"PageRank for Product Image Search"一文中运用的方法是提取 SIFT feature，D.G.Lowe 在"Distinctive Image Features from Scale-Invariant Keypoints"中对这种 feature 进行了详细的说明。

Page 和 Brin 在"The Anatomy of a Large-Scale Hypertextual Web Search Engine"一文中还对搜索引擎的各个环节进行了详细的描述，他们讨论了核心的数据结构，这些数据结构的设计都经过精心的优化并且考虑到了方方面面. 他们讨论了爬虫可能遇到的各种问题以及处理方式. 并且对性能进行了详细的分析. 文中还介绍了词典以及索引的实现，存储的细节和排名函数的设计。

Google 被设计为可扩展的系统能够应对 Web 的爆炸式增长，这都归功于这种精心的设计。

参考文献:
[1] S.Brin and L.Page. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 1998
[2] Shumeet Baluja and Yushi Jing. PageRank for Product Image Search. 2008
[3] S.Brin and L.Page. The PageRank Citation Ranking Bringing Order to the Web. 1998
[4] Taher H. Haveliwala. Effcient Computation of PageRank. 1999
[5] Y.Jing, S.Baluja, and H. Rowley. Canonical image selection from the web. In Proc. 6th International Conference on Image and Video Retrieval(CIVR), 2007
[6] K. Mikolajczyk and C.Schmid. A performance evaluation of local descriptors. IEEE Transaction on Pattern Analysis and Machine Intelligence, 2005

本科毕业论文（设计）外文翻译

大规模超文本 Web 搜索引擎剖析

Sergey Brin and Lawrence Page

{sergey, page}@cs.stanford.edu
Computer Science Department, Stanford University, Stanford, CA 94305

摘要

本文中我们将呈现 Google，这是一个大量利用超文本结构的大规模搜索引擎的原型。Google 的设计目的是高效地爬取和索引 Web 从而产生比现有的系统更让人满意的搜索结果。原型的地址在 http://google.stanford.edu， 其具有至少 2 千 4 百万页面的完整的文本和超链接数据库。搜索引擎对工程师来说是充满挑战的工作。搜索引擎索引成千万上亿的 Web 页面，而涉及的不同的词语也有与此相当的数量，并且每天要处理几千万次的查询。虽然大规模的 Web 搜索引擎十分重要，但是很少有相关的学术研究。此外，由于科技的快速进步和互联网的快速发展，在今天，构建搜索引擎和三年前已大有不同。本文深入描述了我们的大规模 Web 搜索引擎，这是目前我们已知的第一份如此详细的公开描述。除了传统的同数量级的搜索引擎的技术难题，在使用超文本中呈现的附加信息来产生更好的搜索结果的过程中，我们也面临新的技术挑战。怎样构建实际可用的利用超文本中附加信息的大规模系统，本文将对这个问题进行阐述。同时，我们也会关注这个问题：超文本集合不受限制，每个人都可以发布任何他们想要的东西，那么怎样有效地对其进行处理呢？

1. 介绍

Web 给信息检索带来了新的挑战。Web 上的信息量迅速地在增长，同时对 Web 没有什么经验的新用户也在迅速地增加。当用户利用 Web 的超链接在冲浪时，很可能起始点是像雅虎或其他的人工维护的高质量索引。虽然人工维护的列表有效地覆盖流行的主题，但是这种方法很主观，构建和维护成本昂贵，改动速度慢，并且不能覆盖所有深奥的主题。依赖于关键词匹配的自动搜索引擎通常返回太多的低质量匹配，更糟的是，一些做广告的人会采取措施误导自动搜索引擎来吸引人们的注意。我们构建大规模的搜索引擎，设法解决现有系统存在的问题。引擎大量使用了超文本中附加结构信息来提供更高质量的搜索结果。我们把系统命名为

Google，因为这是 googol 或 10^100 的一种普遍拼法，这个名字与我们构建超大规模搜索引擎的目标十分符合。

1.1 Web 搜索引擎——增大规模：1994-2000

搜索引擎技术不得不跟随 Web 前进的脚步。在 1994 年，最早的搜索引擎之一，World Wide Web Worm(WWWW)存有 110，000Web 页面的索引和可访问文档。到了 1997 年的 11 月，顶尖的搜索引擎宣称已经索引了两百万(Web 爬虫角度)至一亿 Web 文档(搜索引擎角度)。可以预计，到 2000 年复杂的 Web 索引数量将会达到 10 亿。与此同时，搜索引擎需要处理的查询数也在以难以置信的速度增长。在 1994 年的 3，4 月份，World Wide Web Worm 平均每天接受 1500 次查询。到 1997 年的 11 月 Altavista 宣称每天接受大概两千万次查询。随着 Web 用户以及自动查询搜索引擎系统数量的增长，到 2000 年，顶尖的搜索引擎很可能每天要处理上亿次的查询。我们系统的目标是解决许多由如此庞大规模的搜索引擎技术带来的可扩展性和质量问题。

1.2 Google——跟上 Web 的脚步

即使是构建适用于当前 Web 情况的搜索引擎也存在很多挑战。 收集 Web 文档需要快速的爬虫技术并且需要保持实时更新。存储空间必须被充分利用来存储索引，有时需要存储文档本身。索引系统必须能够高效地处理上千 GB 的数据，查询必须被快速处理，达到每秒数百至上千次。

随着 Web 的增长，这些任务变得更加艰难。然而，硬件的性能和价格也有显著的改进，这抵消了任务的部分困难。但是，在总体进步的大环境下，有一些值得注意的意外情况，像磁盘寻道时间和操作系统的鲁棒性。在设计 Google 的时候，我们同时考虑了 Web 的增长和技术的变化。Google 被设计成能够处理超大规模的数据集。它有效地利用存储空间来存储索引，数据结构为快速和高效的访问进行了优化。更进一步，我们希望索引和存储文本或 HTML 的成本会较现在降低，这会让 Google 这样的集中系统获得良好的扩展性能。

1.3 设计目标

1.3.1 改进搜索质量

我们的主要目标是改进 Web 搜索引擎的搜索质量。在 1994 年，一些人认为一份完整的索引可以让人很容易地找到任何信息。根据 Best of the Web 1994 -- Navigators 所说，“最好的导航服务应该使得人们在 Web 搜寻任何内容都变得很容易”。然而，1997 年的 Web 已经今非昔比，目前任何使用搜索引擎的人可

以很容易地证明索引的完整并不是决定搜索结果质量的唯一因素。垃圾结果经常会淘汰任何用户感兴趣的结果。实际上，到了 1997 年的 11 月，只有四分之一的商业搜索引擎会查询到自己(搜寻自己的名字会在前十名的结果中返回自己的搜索页)。造成这个问题的一个主要原因是索引中文档的数量已经比过去增长了许多个数量级，但是用户看文档的能力可没有变化。用户还是希望只看前几十条的搜索结果。由于这个原因。随着集合的增大，我们需要高准确率(比如返回的前十几个文档中相关的个数)的工具。实际上，我们对“相关”的定义是只包含非常好的文档，因为会存在成十万上百万的略微相关文档。即使以召回率(系统能够返回的相关文档总数)的牺牲为代价，高准确率也十分重要。最近有一些乐观的发现：更多地使用超文本信息有助于改进搜索和其他应用。尤其是链接结构和链接文本为相关性的判断和质量的过滤提供了很多信息。Google 同时利用了链接结构和链接文本。

1.3.2 搜索引擎学术研究

除了飞速的增长，Web 也越来越倾向于商业化。在 1993 年，1.5%的 Web 服务器域名为.com。到了 1997 年这个数字增长到了 60%。与此同时，搜索引擎已经从学术领域迁移至商业领域。到目前为止，大多数开发搜索引擎的公司都很少披露技术细节。这导致了搜索引擎技术仍然是一种魔术并且是面向广告的。对 Google 而言，我们有一种强烈的意愿，就是把更多的开发过程和解释带到学术领域。另一个重要的设计目标是我们构建的系统能够让可观数量的人群实际使用。可使用性对我们来说很重要，因为我们认为能够利用现代 Web 系统庞大的数据进行一些非常有趣的研究。例如，每天人们会进行上亿次的搜索。但是却很难得到这些数据，因为这些数据被认为是具有商业价值的。

我们的最终目标是构建一个体系，能够支持在大规模的 Web 数据上进行新颖的研究。为了支持新颖研究的进行，Google 以压缩形式存储了所有爬取到的文档。我们设计 Google 的主要目的之一是建立一个能让其他研究者很快加入的环境，处理 Web 的大量信息，来产生通过其他方式很难产生的有趣的结果。在系统被建立起的这段短暂的时间内，已经有一些论文使用 Google 产生的数据库，许多其他的使用者也在研究当中。另一个目标是建立一个像太空实验室一样的环境，研究者甚至学生可以提出自己的意见并且在我们的大规模 Web 数据上进行有趣的实验。

2 系统特性

Google 搜索引擎有两个重要的特性，这使得它能产生高准确率的结果。首先，它充分利用了 Web 的链接结构来计算每一个 Web 页面的质量排名。这个排

名叫做 PageRank，在[Page 98]中有详细的说明。第二，Google 利用链接来改进搜索结果。

2.1 PageRank：给 Web 排序

Web 的引用(链接)图是一种重要的资源，过去的 Web 搜索引擎都没有将之利用起来。我们已经将多达 5 亿 8 千 1 百万的超链接构建成引用图，这是一份具有重要意义的样本。这些图可以快速计算一张网页的"PageRank"，这是一种客观的评估网页的引用重要性的的指标，与人们的主观意见很一致。由于这种一致性，PageRank 是一种绝佳的手段来给基于 Web 关键词的搜索结果排序。对于 Web 页面标题的简单文本匹配搜索，当应用 PageRank 对结果进行排序后也效果也很理想。而对于 Google 的全文搜索，PageRank 也起到了很重要的作用。

2.1.1 计算 PageRank

学术上的文献引用已经被应用到了 Web 上， 主要是对给定页面的引用和反向链接进行计数。这种做法可以近似得到某页面的重要性或者质量。PageRank 对这种思想尽心了扩展，它对于不同的页面的链接会区分对待，并因页面上的连接数来进行归一化。以下是 PageRank 的定义：我们假设页面 A 有 T1 到 Tn 这 n 个页面指向它。参数 d 是介于 0 和 1 之间的阻尼因数。我们通常将 d 设为 0.85。下一节会有更多关于 d 的细节描述。C(A)定义为从页面 A 指出的链接数。页面 A 的 PageRank 定义如下：

PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))

PageRank 可以使用简单的迭代算法来计算，相当于求 Web 归一化链接矩阵的主特征向量。2 千 6 百万页面的 PageRank 在一台中型工作站中几个小时可以计算出来。这其中有许多其他的细节不在本文的讨论范围之内。

2.1.2 直观的道理

PageRank 可以被认为是一种用户行为的模型。我们假设有一个随机的网上冲浪者从一张随机的页面开始点击链接，不会点击后退，但是最终会感到厌烦并且开始浏览另一个新的随机页面。这个冲浪者浏览某网页的概率，就是这张网页的 PageRank。而阻尼参数 d 是在某个页面上这个随机冲浪者会感到厌烦并请求另一个新的随机页面的概率。一项重要的改变是只对单个或一组页面添加阻尼因数。这样做更人性化并且使得故意误导系统来获得更高排名的做法几乎不可能成功。我们对 PageRank 有一些其他的扩展，见[Page 98]。另一个直观的道理是一张页面如

果有很多页面指向或者指向它的一些页面具有高 PageRank，它那就应该具有高的 PageRank。显而易见，在 Web 上被许多地方引用的页面值得一看。同时，如果一张页面只有一个引用但是是被 Yahoo！首页引用的话一般也是值得一看的。如果某张页面并非高质量页面或者是一个死链，Yahoo！的主页就没什么可能会链向它。PageRank 同时考虑了这些因素并且可以处理 Web 链接结构中权重的递归传递问题。

2.2 链接文本

链接文本在搜索引擎中会被特殊对待。大多数搜索引擎只把链接文本和链接所在的页面关联起来。除此之外，我们将其与指向的页面也关联起来。这样做有几个优点：首先，链接文本提供比页面自身更准确的信息。第二，对像图片，程序和数据库这种基于文本的搜索引擎无法索引的文档也可能拿到链接文本。这使得返回没有被爬虫实际爬取到的页面成为可能。这里需要注意的是没有被爬下的页面可能导致问题，因为这些页面在返回给用户之前没有检测过有效性。在这种情况下，搜索引擎甚至可能返回有链接指向但是根本不存在的页面。然而，因为可以对结果进行排序，所以这种特殊的问题很少发生。

利用链接文本这种方法在 World Wide Web Worm 中实现，主要因为这种方法可以帮助搜索非文本信息，并且可以利用较少下载下来的文档来拓展搜索覆盖率。我们使用链接文本主要是由于它可以帮助我们提供更好的搜索结果。高效的使用链接文本是一个技术难题，因为有大量的数据必须处理。在我们目前抓取 2 千 4 百万的页面中，索引的链接有 2 亿 5 千 9 百万。

2.3 其他特性

除了 PageRank 和对链接文本的使用之外，Google 还有一些其他特性。首先，Google 具有所有点击的位置信息并在搜索中广泛应用。第二，Google 关注一些视觉表示的细节例如字体的大小。更大更粗的字体会拥有更高的权重。第三，完整的原 HTML 页面全部存储下来。

3 相关工作

关于搜索的研究历史很短。World Wide Web Worm(WWWW)是最早的 Web 搜索引擎之一。后来又产生了一些其他用于学术研究的搜索引擎，这些引擎中很多现在都成为了上市公司。虽然 Web 在快速增长并且搜索引擎十分重要，但是与之相关的研究却很少。就像 Michael Mauldin 所说，“不同的服务都对数据库的细节守口如瓶”。然而，人们在搜索引擎上做了许多工作。尤其具有代表性的

是由已经存在的商业搜索引擎得到的结果进行后期处理来得到新的结果，或者小规模的个性化的搜索引擎。最后，有关信息检索的研究有很多，尤其是对于有组织集合。在下面两章中，我们讨论一些领域，这些领域我们需要进行拓展，以便在 Web 上获得更佳的性能。

3.1 信息检索

信息检索系统的相关工作在很多年前就已经广泛开展。然而，大多数信息检索系统建立在小规模的同质集合上，例如科学论文或者特定主题的新闻。实际上，信息检索的主要基准，文本检索会议，基于相当小的易控集合。所谓的大规模的标准也只有 20G，而我们爬取的 2 千 4 百万网页具有 147GB。在 TREC 上效果良好的方法在 Web 上不会产生很好的效果。例如，标准的向量空间模型会根据给定查询和词频向量尝试返回与查询最接近的文档。在 Web 上，这种策略经常会返回非常短的文档，只含有查询词加上一些其他内容。比如，我们看到过搜索引擎对“Bill Clinton”这个查询返回的结果只含有“Bill Cliton Sucks”和图片。一些人主张在 Web 上用户应该更清楚的指明想要的内容并且在查询中加入更多的词。我们强烈反对这种观点。如果用户查询像“Bill Clinton”这样的词，他们理应得到合理的结果，因为关于这个话题有大量的高质量信息。由这些例子可知，标准的信息检索方法为了能够有效的应用于 Web 需要进行扩展。

3.2 Web 与有组织集合的不同

Web 是一个庞大的无组织异质文档集合。文档间的内在差异很大，能够得到的外部元信息也各有不同。例如，文档本身的语言（人类语言或程序），词汇（email 地址，链接，邮政编码，电话号码，产品编号），类型或格式（text，HTML，PDF，图像，音频）都会存在内在的差异，甚至可能是机器产生的（日志文件或数据库的输出）。另一方面，我们把外部的元信息定义为一种可以推测文档内容的信息，但是不是文档的一部分。外部信息会含有来源的权威性，更新频率，质量，受欢迎程度和引用。不仅外部信息源的信息可能不同，需要考虑的事情的数量级也不同。例如，把一个像 Yahoo 这样的每天接受上千万次访问的主流主页和一篇十年才有人看一次的生涩的历史性文章相比。很明显，这两种信息在搜索引擎中应该区分对待。

另一个 Web 与传统的有组织集合重大的不同是 Web 无法控制人们放在上面的内容。这种自由性对搜索引擎有很大的影响，搜索引擎必须处理一些为了盈利故意误导搜索引擎的公司的行为，这是一个严重的问题。这个问题传统的封闭式信息检索系统是无法解决的。甚至有很多公司为了盈利专门研究怎样操纵搜索引擎。

4 系统剖析

首先，我们会对系统的体系进行概括的介绍。之后是一些对重要的数据结构的深层次讨论。最后，对主要的应用：爬虫，索引系统，查询系统进行深入的检验。

4.1 Google 体系综述

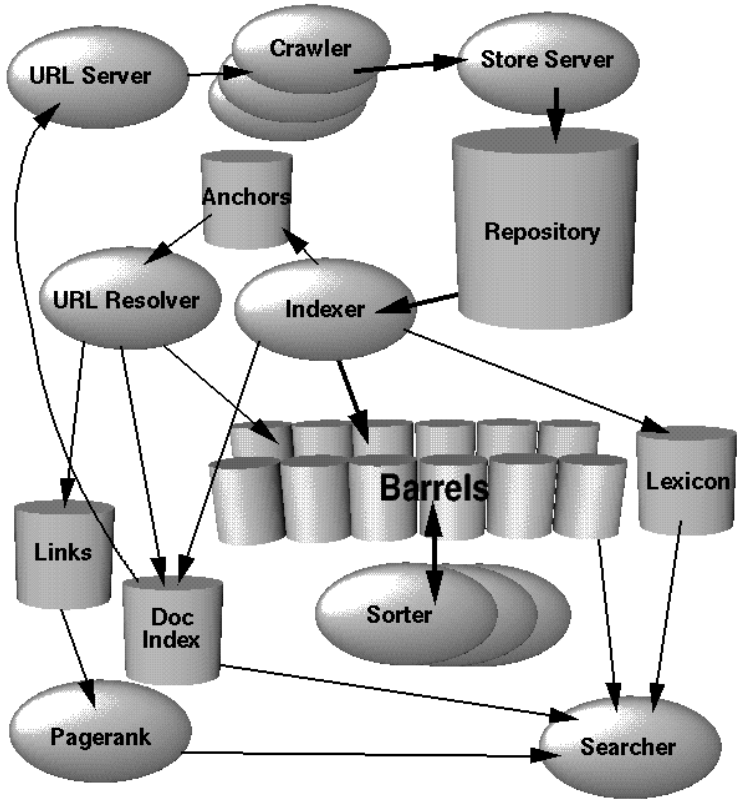


图 1. Google 体系概览

在这一节，我们会对系统进行概括的描述，图 1 描述了整个系统是如何工作的。接下来的部分我们会讨论本节没有讨论的应用和数据结构。Google 大多数的程序为了效率考虑由 C 或 C++实现并且可以在 Solaris 或 Linux 上运行。

Google 的 Web 爬取工作（下载 Web 页面）由一些分布式的爬虫程序完成。一台 URL 服务器把 URL 分发给爬虫。爬取的网页随后被传输到存储服务器。存储服务器会把 Web 页面进行压缩并存储在数据仓库中。每张 Web 网页都对应一个 ID 号叫做 docID，这是在一个新的 URL 在 Web 页面中解析出来时指定的。索引程序和排序程序负责构建索引。索引程序需要实现很多功能，首先要从数据仓库中读取数据，并将文档进行解压缩，之后进行解析。每个文档被转化为一种叫做 hits 的存储格式，hits 记录词，词在文档中出现的位置，字体大小的近似值以及大

小写情况。然后索引程序把这些 hits 分成几份，构建部分排序的前向索引。索引程序还有另一个重要的功能。它会解析 Web 页面中的所有链接并且把与之相关的重要信息存储在链接文件中。这个文件含有充足的信息来判断每个链接的指向以及相应的链接文本。

URL 解析器读取链接文件并把相对 URL 转化为绝对 URL 然后依次转化为 docID。解析器把链接文本存储在前向索引中，并把它与所指向的 docID 关联起来。解析器同时会建立由 docID 对组成的链接数据库。这个连接数据库用来计算所有文档的 PageRank。

排序程序接受根据 docID 排序的每份 hits，根据 wordID 重排并产生倒排索引。这项工作在合适的地方进行以保证只会占用很少的临时空间。排序程序同时产生 wordID 列表以及在倒排索引中的偏移量。一种叫做 DumpLexicon 的程序接受索引程序产生的词典把这些列表进行整理并产生新的词典提供给查询系统使用。查询系统由一台 Web 服务器启动，同时使用 DumpLexicon 构建的词典，倒排索引和 PageRank 来处理查询。

4.2 主要数据结构

Google 为了能够以低成本支持爬虫，索引系统，查询系统在大规模的文档上的运行，优化了数据结构。虽然 CPU 和大容量输入输出率近年来得到了大幅的提高，但是一次磁盘寻道时间仍然需要花费大约 10ms 才能完成。Google 被设计为尽可能的避免寻道，这方面的考虑对 Google 的数据结构的设计产生了很深的影响。

4.2.1 Bigfiles

Bigfiles 是 64 位寻址的跨越多种文件系统的虚拟文件。可以自动处理多种文件系统的分配。Bigfiles 包同时可以处理申请和释放文件描述符，因为操作系统并没有提供能够满足我们需求的功能。 Bigfiles 还支持初级的压缩功能可供选择。

4.2.2 仓库(Repository)

仓库包含每张 Web 页面的完整的 HTML，每张页面使用 zlib(见 RFC1950)压缩。压缩技术的选择要求在速度和压缩比上进行权衡。相比较于 bzip 在压缩比上的优势，我们还是倾向于 zlib 的速度优势。bzip 的压缩比接近 4 比 1 而 zlib 的是 3 比 1。仓库中文档一个接一个的存储，每条文档以 docID，长度和 URL 开头，

如图 2 所示。访问仓库不需要使用其他的数据结构。这有助于数据的一致性并使开发变得简单；我们只需要仓库和列出爬虫错误的文件就可以重新构建所有其他的数据结构。

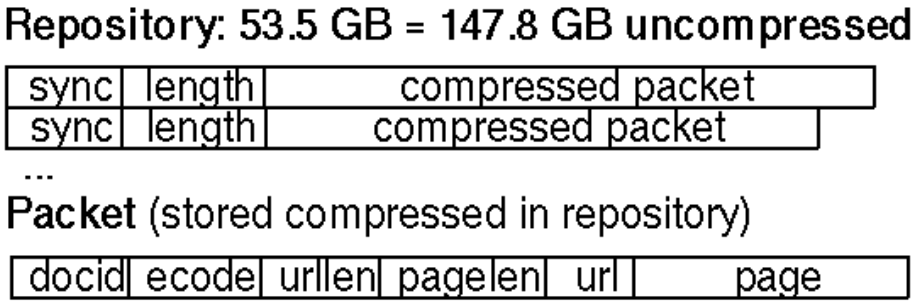


图 2. 仓库数据结构

4.2.3 文档索引

文档索引保留每条文档的信息。文档索引是由 docID 排序的固定宽度的 ISAM(索引序列访问模型)索引。每条存储的信息包括当前的文档状态，指向仓库的指针，文档的校验和，和许多种统计信息。如果文档已经被爬取，文档索引还会含有一个指针指向一个叫做 docinfo 的不定长文件，这个文件中存有文档的 URL 和标题。否则这个指针指向只含有 URL 的 URL 列表。采用这种设计是因为可以拥有紧凑的数据结构，并且可以在一次查询中只通过一次磁盘寻道就可以获得数据。

另外，存在一类文件用来把 URL 对应到 docID。这种文件是以校验和排序的一列 URL 校验和，带有相应的 docID。为了找到指定 URL 的 docID，首先计算 URL 的校验和，然后在校验和列表上进行二分搜索就可以找到对应的 docID。通过与这个文件作何并，URL 可以批量转化为 docID。这是 URL 解析器把 URL 转化为 docID 使用的技术。批量升级模式很重要，因为如果没有这种模式我们必须对每个链接进行一次磁盘寻道，如果是这样，在一块磁盘上对我们的 3 亿 2 千 2 百万数据集进行处理需要超过一个月的时间。

4.2.4 词典

词典有几种不同的形式。与之前的系统相比，一个重要的变化就是词典可以以能够接受的代价存于内存中。按照当前的实现我们可以把词典放在拥有 256MB 内存的机器上。当前的词典包含 1 千 4 百万个单词（但是一些稀有词没有被加入词典）。词典由两部分组成，一张单词列表（连接在一起，以 null 分隔）和一张指针哈希表。为了其他一些功能，单词列表含有一些辅助信息，详细的解释不在本文的讨论范围之内。

4.2.5 Hit Lists

hit lists 含有某个词在某篇文档中出现的相关信息，包括位置，字体和大小写信息。Hit lists 占用了正向索引和倒排索引的大多数空间。正因如此，尽可能高效的表示 Hit lists 十分重要。我们考虑了几种方法来编码位置，字体和大小写信息，简单编码（三个整数），压缩编码（bit 手动优化申请）和霍夫曼编码。最后我们选择了手动 bit 优化压缩编码因为这比简单编码占用少的多的空间，比霍夫曼编码进行更少的位操作。图 3 展示了 hits 的细节。我们的压缩编码对每个 hit 使用两个字节，有两种类型的 hits：特殊的和普通的。特殊的 hits 包含 URL，标题，链接文本和 meta 标签中出现的 hits，普通的包含其他所有的 hits。普通的 hits 包含一个大小写位，字体大小和 12 位的词在文档中的位置（所有大于 4095 的位置标记为 4096）。字体大小使用相对于文档其他部分的相对大小用三位表示（实际只有 7 个值能用因为 111 用来标记特殊 hits）。特殊的 hit 包含一个大小写位，字体位设为 7 表示这是一个特殊的 hit，4 位编码特殊 hit 的类型，8 位来记录位置。对于链接文本的 hits，8 位的位置信息被分为 4 位的链接文本位置和 4 位的链接文本所在文档的 docID 的哈希。如果某个词没有足够的链接文本，短语查询就会受到限制。我们打算改进 hits 存储的方式，以便解决 docID 哈希字段的位数不足的问题。我们使用相对大小字体因为在搜索时，人们不会只因为字体大小就把两篇完全相同的文档区分对待。

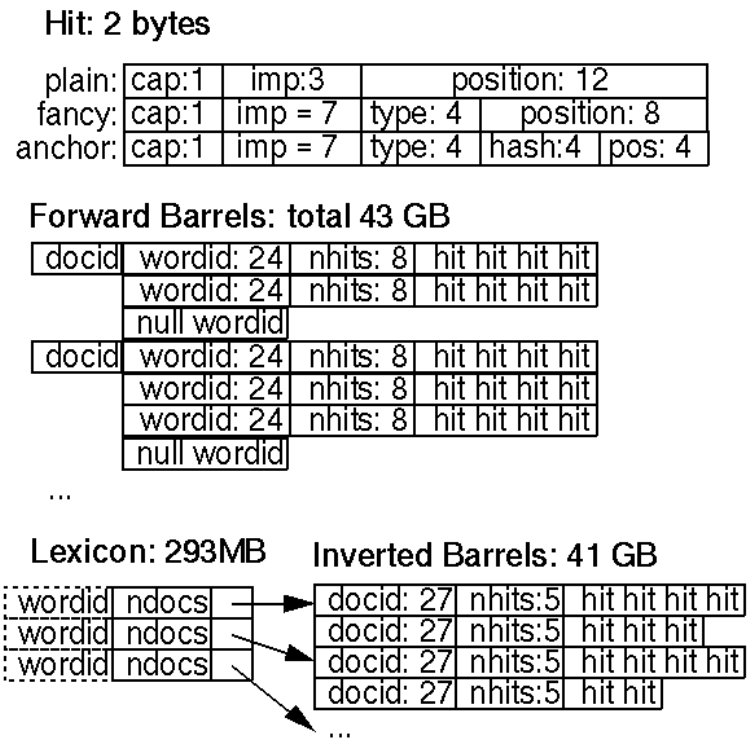


图 3. 正向索引，倒排索引和词典

在 hit 的前面存有 hit 的长度。为了节省空间，hit 列表的长度和 wordID 一起存储在前向索引中，docID 存储在倒排索引中。这把他们分别限制在 8 位和 5 位（从 wordID 中借用 8 个 bit 采用了一些技巧）。如果长度超过这些 bit 的范围，就在这些位上使用转义码，然后下两个字节包含实际长度。

4.2.6 正向索引

正向索引其实已经部分排序。排序过程是在一些(我们使用 64 个)桶中进行的。每个桶处理一定范围的 wordID 对应的正向索引。如果一条文档含有对应到某个桶的词，那对应的 docID 被记录到桶中，然后是对应这些词的 wordID 列表，同时带有 hit 列表。这种模式因为重复记录了 docID 会略微增加存储空间但是桶的数量合理的话影响不是很大，但是节省了很多的时间，并且排序程序最后索引阶段的编码复杂度得到了有效地降低。此外，我们并不存储实际的 wordID，而是存储每个 wordID 与当前桶中最小的 wordID 的相对差值。这样我们就能在未排序的桶中对只使用 24 位来存储 wordID，把剩余的 8 位用来存储 hit 列表的长度。

4.2.7 倒排索引

倒排索引与正向索引拥有相同的桶个数，不同的是倒排索引已经被排序程序处理过。对于每个合法的 wordID，词典含有一个指针指向 wordID 对应的桶。这个指针指向带有相应 hit 列表的 doc 列表。doc 列表含有所有词在所有文档中出现的信息。

一个重要的问题是 docID 在 doc 列表中应该以什么顺序排列。一种简单的解决方式是按照 docID 排序存储。这种方式支持对多词查询的快速 doc 列表合并。另一种方式是按照词在每条文档中出现的次数排序。这种方式令单词查询很简单，多词查询效果也不错。然而，合并对这种方式来说更困难。同时，采用这种方式一旦排序函数发生变化需要重新构建索引，这也会给开发带来更大难度。我们选择了一种折中的方式，保留两个倒排桶集合，一个集合包含标题和链接文本的 hit 列表，另一个集合包含所有的 hit 列表。这样，我们可以首先检查第一个桶集合，如果匹配数量不足我们再检查更大的那个。

4.3 Web 爬取

运行 Web 爬虫是充满挑战的工作。会存在很多性能问题，可靠性问题甚至更重要的是社会问题。爬虫是最容易出问题的系统因为它需要与成千上万的 Web 服务器和域名服务器打交道，这都超出了系统自身的控制范围。

为了能够处理上十亿的 Web 页面，Google 拥有快速的分布式爬虫系统。一台 URL 服务器为一部分爬虫机（一般是 3 台）提供 URL 列表。URL 服务器端程序和爬虫机程序都由 Python 实现。每台爬虫机同时大约建立 300 个连接。这对快速获取 Web 页面是很必要的。系统使用四台爬虫机最快每秒可以爬取超过 100 张 Web 页面。大约每秒 600K 数据。主要的性能压力来自 DNS 查询。每台爬虫维护自己的 DNS 缓存，这样就不需要在爬取每条文档时都进行 DNS 的查询。这几百个连接可能有几种不同的状态：查询 DNS，连接主机，发送请求，接收请求。这些因素导致爬虫是系统中十分复杂的组件。爬虫使用异步 IO 管理事件，使用一些队列来转移抓取页面的状态。

结果显示爬虫会连接多于 50 万的服务器，并产生数千万的访问记录，这给我们带来了很多的电子邮件和电话。因为网络上的人非常多，所以总是会有一些人不知道爬虫是什么，因为这是他们第一次见到爬虫。几乎每天我们都会收到这样的邮件，“哇，你在我的网站浏览了好多页面，你认为我们的网站怎么样？”还有一些人不知道机器人排除协议，他们在网页上写着“本页面受版权保护，禁止爬取”，他们认为这样就可以保护他们的页面不被索引。这对于 Web 爬虫程序来说无疑是难以识别的。因为涉及的数据量规模很大，总会发生意料之外的情况。举例来说，我们的系统会尝试爬取在线游戏。游戏中会有很多垃圾信息，虽然修改这个问题很简单，但是在发现这个问题的时候我们已经爬取了几千万的页面。因为 Web 页面和服务器多种多样，所以检验爬虫的唯一方法就是在大部分的 Internet 上跑过。总是有很多隐晦的问题会出现，这些问题可能只在整个 Web 的一张页面中出现，而导致爬虫崩溃，更糟的是，会导致不可预知的或不正确的行为。访问大部分 Internet 的系统需要设计的非常鲁棒并且需要经过仔细的测试。因为像爬虫这样的大型复杂系统总是会出现问题，所以当这些问题出现的时候，我们需要花很大的精力来阅读 email 并解决这些问题。

4.4 索引 Web

解析——任何在整个 Web 上运行的解析程序必须处理相当多的可能遇到的错误。其中包括 HTML 标签的拼写错误，标签中含有几千字节的 0，非 ASCII 字符，HTML 标签嵌套几百层深，还有各种各样具有创意的错误挑战人们的想象力。为了速度最大化，我们使用具备自己的栈的词法分析程序而不选用 YACC 的上下文无关解析器。开发既高效有鲁棒的解析器需要相当大量的工作。把文档索引至桶中——所有文档在被解析之后，被编码到桶中。每个词被转换为 docID，这需要使用一张内存中的哈希表——词典。为词典增加新内容会记录在日志文件中。一旦词被转换为 wordID，它们在当前文档中的出现被转化为 hit 列表然后被写在前向索引桶中。索引阶段并行化的主要难题是词典需要共享。我们没有采用分享内存的方法，而是采用对所有不在基本词典中出现的词写一条记录，基本词典固定在 1 千 4 百万个词。通过这种方式，多个索引程序可以并行执行， 多出

的词的小记录文件可以由最终的索引程序处理。

排序——为了构建倒排索引，排序程序接受每个正向索引桶并按照 wordID 对其进行排序，产生标题和链接文本的倒排索引桶和全文本倒排索引桶。这个过程一次只处理一个桶，所以需要很少的临时存储空间。同样的，我们使用尽可能多的机器来运行多个排序程序使得排序过程并行化，同时处理不同的桶。因为桶不能被放进内存，排序程序根据 wordID 和 docID 把桶分为能放入内存的几部分，再把每一部分载入内存进行排序最后把内容写进短倒排桶和全倒排桶。

4.5 搜索

搜索程序的目的是高效的提供高质量的搜索。大多数大规模的商业搜索引擎都在效率方面有很大的进展。因此，我们在研究中更关注搜索质量，但是我们相信通过一些努力我们也可以达到商业搜索引擎的性能。Google 查询的评估过程如图 4 所示。

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.
   Sort the documents that have matched by rank and return the top k.

图 4. Google 查询评估过程

为了限制响应时间，一旦匹配到了一定数量（目前是 4 万）的文档，搜索程序自动跳至步骤 8，如图 4 所示。这意味着返回的结果是次优的。我们目前正在研究其他的方法来解决这个问题。曾经我们按照 PageRank 来排序 hits，似乎对这个问题有所改进。

4.5.1 排名系统

Google 比传统的搜索引擎保留了更多有关 Web 文档的信息。每个 hitlist 包含位置，字体，和大小写信息。此外，我们将链接文本和文档的 PageRank 信息纳入 hits。把所有的信息综合起来考虑排名是很困难的事。我们设计的排名函数使得没有某种因素会具有十分大的影响。首先，考虑最简单的情况——一个单词查询。为了给一条单词查询的文档排名，Google 查询这个词的文档 hit 列。

Google 的 hit 有几种不同的类型（标题，链接文本，URL，纯文本大字体，纯文本小字体，...），每一种都由自己的权重。权重以类型索引组成一个向量。Google 对每种 hit 列表中命中的数量进行计数。然后把技术转化为带权计数。带权计数一开始随着计数增多线性增长但是增长率很快变慢，当计数超过一定值后就不会再起作用。我们对带权计数向量和类型权重向量求点积计算出每个文档的 IR 分值。最后我们用这个 IR 分值和 PageRank 综合计算出一个文档的最终分值。

对于多词搜索来说，情况更加复杂。现在多个 hit 列表必须被一起扫描，这样在文档中紧凑出现的词会比分开出现的权重更大。多个 hitlist 被匹配这样相邻的 hits 才能匹配。对每个匹配的 hits 集合计算相似性。相似性基于 hits 在文档（或链接文本）中的距离，距离被分为十类，从短语匹配到“不接近”。不仅需要对每种类型的 hit 计数，同时也要对每种相似度进行计算。每种类型和相似度对有一个权重。计数被转化为带权的然后和类型相似度对的权重向量计算点积，得出 IR 分数。这些数字和矩阵在一种特定的 debug 模式下都可以随搜索结果显示出来。这些显示出来的数据对开发排名系统来说十分有用。

4.5.2 反馈

排名函数有很多参数，像类型权重，类型相似权重。计算这些参数的值有些像魔法。为了做到这点，我们在搜索引擎中设置了反馈机制。受信任的用户可以选择对搜索结果进行评价。反馈的信息会得到保存。当我们需要修改排名函数的时候，我们可以看到这种改变对之前的排名造成何种影响。虽然还远远称不上完美，但是这给我们一些办法能够了解对排名函数的修改是怎样影响搜索结果的。

5 结果和性能

衡量搜索引擎的最重要的标准就是搜索结果的质量。一份完整的用户评估不在本文的范围之内，但是我们自身对 Google 的体验是，对于大多数搜索 Google 都能够产生比当前主流的商业搜索引擎更好的搜索结果。作为一个解释 PageRank，链接文本和相似性是如何使用的例子，图 5(译者注：原文中写的是图 4，应该是笔误)展示了 Google 对查询“Bill Clinton”的搜索结果。这些结果表现了 Google 的一些特性。这些结果由服务器聚类。这对筛选结果集帮助很大。很多结果来自 whitehouse.gov 域名，这是人们搜索这个词时很可能期望的搜索结果。目前，大多数主流的搜索引擎不会返回 whitehouse.gov 的任何结果，正确的结果也要少的多。我们注意到第一条搜索结果是没有标题的。这是因为它并没有被爬取。Google 使用链接文本判断查询结果的好坏。相似的，第五条结果是 email 地址，当然也是无法爬取的。这也是链接文本起的作用。

图 5. *Google* 的结果示例

所有的结果都是高质量的页面，并且最后经过检查没有死链。这主要是因为这些页面都具有高分的 PageRank。PageRank 由带有红色条状图的百分比表示。最后，没有结果只含有 Bill 而不含有 Clinton，也没有结果只含有 Clinton 而不含有 Bill。这是因为我们很重视词出现的相似性。当然，要对搜索引擎的质量进行测试需要一份用户调研和结果分析，但是我们这里没有空间来写这些。取而代之的是，我们邀请读者亲自去 http://google.stanford.edu 体验一下 Google。

5.1 存储需求

除了搜索质量，Google 的设计能够随着 Web 的增长有效地扩展。这其中一个方面就是高效利用存储。表 1 展示了 Google 的一些统计数据和存储需求。由于仓库压缩后总体积大约 53GB，只比存有的总数据的三分之一多一些。以当前的磁盘价格来讲，仓库用来存储有用的数据是相对便宜的资源。更重要的是，搜索引

擎需要的总数据量与之相当，大约 55GB。此外，大多数查询只使用短倒排索引就可以得到解答。采用更先进的文档索引压缩技术，高质量的 Web 搜索引擎可以在一台新的 PC 机上运行，只占用 7GB 磁盘。

Storage Statistics	
Total Size of Fetched Pages	147.8 GB
Compressed Repository	53.5 GB
Short Inverted Index	4.1 GB
Full Inverted Index	37.2 GB
Lexicon	293 MB
Temporary Anchor Data (not in total)	6.6 GB
Document Index Incl. Variable Width Data	9.7 GB
Links Database	3.9 GB
Total Without Repository	**55.2 GB**
Total With Repository	**108.7 GB**

Web Page Statistics	
Number of Web Pages Fetched	24 million
Number of Urls Seen	76.5 million
Number of Email Addresses	1.7 million
Number of 404's	1.6 million

表 1. 统计数据

5.2 系统性能

对搜索引擎来说，高效的爬虫和索引很重要。这样信息可以实时更新，主要的系统变动也可以较快的得到测试。Google 主要的操作有爬虫，索引和排序。测量爬虫需要的总时间并不容易，因为磁盘空间满，域名服务器崩溃或者任何其他的问题都会使系统停止。我们总共花了大概 9 天下载了 2 千 6 百万的页面（包括错误的）。然而，一旦系统能够流畅的运行，速度要快得多，下载最后的 1 千 1 百万页面只用了 63 个小时，平均每天超过 4 百万页面或者 48.5 张每秒。我们同时运行索引程序和爬虫程序。索引程序比爬虫运行速度略快。这主要是因为我们花了足够的时间来优化索引程序使之不会成为瓶颈。这些优化包括文档索引的批量更新和重要的数据结构在磁盘上的存储。索引程序每秒大概处理 54 张页面。排序程序可以完全并行执行；使用四台机器，整个排序过程需要大约 24 小时。

5.3 搜索性能

到目前为止，改进搜索的性能不是我们主要的关注点。当前版本的 Google 处理大多数查询在 1 至 10 秒之间。时间主要花在网络文件系统的磁盘 IO（因为磁盘是分布在几台机器上的）。此外，Google 对查询词缓存，常用词子索引和其他常用的优化点上没有进行任何优化。我们倾向于通过分布式，软硬件和算法上的改进来加快 Google 的速度。我们的目标是每秒可以处理几百条查询。表 2 列出了当前版本的 Google 对示例查询的处理时间，以及经过缓存 IO 加速过的时间。

	Initial Query		Same Query Repeated (IO mostly cached)	
Query	CPU Time (s)	Total Time (s)	CPU Time (s)	Total Time (s)
al gore	0.09	2.13	0.06	0.06
vice president	1.77	3.84	1.66	1.80
hard disks	0.25	4.86	0.20	0.24
search engines	1.31	9.63	1.16	1.16

表 2. 搜索时间

6 总结

Google 被设计为可扩展的搜索引擎。主要的目的是随着互联网的飞速发展提供高质量的搜索结果。Google 采用了一系列的技术改进搜索质量，其中包括 page rank，链接文本和相似信息判断。此外，Google 是一个完整的架构，可以用来收集 Web 网页，进行索引，并进行搜索。

6.1 未来的工作

大规模的搜索引擎是很复杂的系统，还有很多工作需要去完成。我们的短期目标是改进搜索效率并且扩容至 1 亿 web 页面。一些简单的效率改进包括查询词缓存，智能磁盘申请和子索引。另一个需要很多研究的领域是更新。我们必须有智能的算法来决定哪些就页面应该重新爬取，哪些新页面应该被爬取。[Cho98]已经开展了这方面的工作。一个有前途的研究领域是使用代理缓存构建搜索数据库，因为

这是需求驱动的。然而，其他的特性刚刚开始探索，像相关性反馈和聚类（Google 目前支持简单的基于主机名的聚类）。我们还计划支持用户上下文信息（像用户的位置信息）和结果摘要。我们同时致力于扩展链接结构和链接文本的使用。简单的实验表明可以通过增加用户主页和书签的权重的方法令 PageRank 个性化。至于链接文本，我们正在尝试将链接周围的文本加入链接文本中。Web 搜索引擎对研究的想法来说拥有很丰富的资源。我们有太多东西想在这里列出，所以在未来，这一章节估计不会变短。

6.2 高质量搜索

如今 web 搜索引擎用户面临的最重要的问题是他们得到的搜索结果的质量。然而结果往往很滑稽，与用户期望的毫不相干，他们会感到很沮丧，并且浪费了宝贵的时间。举例来说，在最流行的商业搜索引擎搜索“Bill Clinton”，第一个搜索结果是 Bill Clinton Joke of the Day: April 14, 1997。Google 被设计为能够随着 Web 的迅速发展提供越来越高质量的搜索，让信息可以被轻松的找到。为了实现这个目标，Google 大量利用了超文本信息，包扩链接结构和链接文本。Google 还使用了相似信息判断和字体信息。虽然衡量搜索引擎是很难的一件事，但是我们主观认为 Google 会比目前的商业搜索引擎返回更高质量的搜索结果。通过 PageRank 的链接结构分析，Google 可以对 web 页面的质量进行评估。使用链接文本来描述链接指向的页面帮助搜索引擎返回相关的（某种程度上说高质量）结果。最后，我们使用信息相似性判断，这有助于增加很多查询结果的相关性。

6.3 可扩展架构

除了搜索质量，Google 设计为可扩展系统。Google 必须高效利用时间和空间，在处理整个 Web 时，不变因素十分重要。在实现 Google 的时候，我们在以下地方遇到了瓶颈，CPU，内存访问，内存容量，磁盘寻道，磁盘吞吐量和网络 IO。Google 在不同的操作中设法克服了其中的一些瓶颈。Google 的主要数据结构充分利用了可用的存储空间。此外，爬虫，索引和排序操作足够高效可以构建 Web 大部分的索引——在一周之内构建 2 千 4 百万页面。我们期望在一个月之内构建 1 亿页面的索引。

6.4 研究工具

Google 不仅是高质量的搜索引擎，还是研究工具。Google 收集的数据已经让很多论文在会议中得到发表，并且更多的论文正在研究中。最近[Abiteboul 97]研究表明如果没有 Web，很多问题无法得到解决。这意味着 Google(或者相似的

系统)不仅是有价值的研究工具，同时对很多领域的应用也是必须品。我们希望 Google 能够成为全世界搜索者和研究者的资源，并且引起下一代搜索引擎技术的革命。

7 鸣谢

Scott Hassan 和 Alan Steremberg 在 Google 的开发中起到了决定性的作用。他们天才的贡献是无可取代的，我们对他们十分感谢。我们还要感谢 Hector Garcia-Molina, Rajeev Motwani, Jeff Ullman 和 Terry Winograd 以及整个 WebBase 组，感谢他们的支持和富有洞察力的讨论。最后我们要感谢我们的设备赞助商 IBM，Intel，Sun 和我们的投资者的慷慨支持。以上研究作为斯坦福综合数字图书馆项目的一部分，由 IRI-9411306 合作协议的国家自然科学基金会支持。为合作协议提供资金的还有 DAPRA，NASA，Interval Research 以及斯坦福综合数字图书馆项目的产业合作人。

参考文献

- Best of the Web 1994 -- Navigators http://botw.org/1994/awards/navigators.html
- Bill Clinton Joke of the Day: April 14, 1997. http://www.io.com/~cjburke/clinton/970414.html.
- Bzip2 Homepage http://www.muraroa.demon.co.uk/
- Google Search Engine http://google.stanford.edu/
- Harvest http://harvest.transarc.com/
- Mauldin, Michael L. Lycos Design Choices in an Internet Search Service, IEEE Expert Interview http://www.computer.org/pubs/expert/1997/trends/x1008/mauldin.htm
- The Effect of Cellular Phone Use Upon Driver Attention http://www.webfirst.com/aaa/text/cell/cell0toc.htm
- Search Engine Watch http://www.searchenginewatch.com/
- RFC 1950 (zlib) ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html
- Robots Exclusion Protocol: http://info.webcrawler.com/mak/projects/robots/exclusion.htm
- Web Growth Summary: http://www.mit.edu/people/mkgray/net/web-growth-summary.html
- Yahoo! http://www.yahoo.com/

- [Abiteboul 97] Serge Abiteboul and Victor Vianu, *Queries and Computation on the Web*. Proceedings of the International Conference on Database Theory. Delphi, Greece 1997.
- [Bagdikian 97] Ben H. Bagdikian. *The Media Monopoly*. 5th Edition. Publisher: Beacon, ISBN: 0807061557
- [Chakrabarti 98] S.Chakrabarti, B.Dom, D.Gibson, J.Kleinberg, P. Raghavan and S. Rajagopalan. *Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text.* Seventh International Web Conference (WWW 98). Brisbane, Australia, April 14-18, 1998.
- [Cho 98] Junghoo Cho, Hector Garcia-Molina, Lawrence Page. *Efficient Crawling Through URL Ordering.* Seventh International Web Conference (WWW 98). Brisbane, Australia, April 14-18, 1998.
- [Gravano 94] Luis Gravano, Hector Garcia-Molina, and A. Tomasic. *The Effectiveness of GlOSS for the Text-Database Discovery Problem.* Proc. of the 1994 ACM SIGMOD International Conference On Management Of Data, 1994.
- [Kleinberg 98] Jon Kleinberg, *Authoritative Sources in a Hyperlinked Environment*, Proc. ACM-SIAM Symposium on Discrete Algorithms, 1998.
- [Marchiori 97] Massimo Marchiori. *The Quest for Correct Information on the Web: Hyper Search Engines.* The Sixth International WWW Conference (WWW 97). Santa Clara, USA, April 7-11, 1997.
- [McBryan 94] Oliver A. McBryan. GENVL and *WWWW: Tools for Taming the Web. First International Conference on the World Wide Web.* CERN, Geneva (Switzerland), May 25-26-27 1994. http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps
- [Page 98] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Manuscript in progress. http://google.stanford.edu/~backrub/pageranksub.ps
- [Pinkerton 94] Brian Pinkerton, *Finding What People Want: Experiences with the WebCrawler.* The Second International WWW Conference Chicago, USA, October 17-20, 1994. http://info.webcrawler.com/bp/WWW94.html
- [Spertus 97] Ellen Spertus. *ParaSite: Mining Structural Information on the Web.* The Sixth International WWW Conference (WWW 97). Santa Clara, USA, April 7-11, 1997.
- [TREC 96] *Proceedings of the fifth Text REtrieval Conference (TREC-5).* Gaithersburg, Maryland, November 20-22, 1996. Publisher: Department of Commerce, National Institute of Standards and Technology. Editors: D. K. Harman and E. M. Voorhees. Full text at: http://trec.nist.gov/

- [Witten 94] Ian H Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* New York: Van Nostrand Reinhold, 1994.
- [Weiss 96] Ron Weiss, Bienvenido Velez, Mark A. Sheldon, Chanathip Manprempre, Peter Szilagyi, Andrzej Duda, and David K. Gifford. *HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering.* Proceedings of the 7th ACM Conference on Hypertext. New York, 1996.