工学硕士学位论文

分布式网络爬虫技术的研究与实现

苏旋

哈尔滨工业大学

2006 年 6 月

国内图书分类号：TP391.3
国际图书分类号：681.37

工学硕士学位论文

分布式网络爬虫技术的研究与实现

硕士研究生：苏旋
导　　师：王晓龙　　教　授
副 导 师：徐志明　　副教授
申请学位：工学硕士
学科、专业：计算机科学与技术
所在单位：计算机科学与技术学院
答辩日期：2006 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index TP391.3
U.D.C:681.37

Dissertation for the Master Degree in Engineering

THE RESEARCH, IMPLEMENT ON TECHNOLOGY OF DISTRIBUTED WEB CRAWLER

Candidate:	Su Xuan
Supervisor:	**Prof. Wang Xiaolong**
Vice-Supervisor:	**Associate Prof. Xu Zhiming**
Academic Degree Applied for:	**Master of Engineering**
Specialty:	**Computer Science and Technology**
Affiliation:	**School of Computer Science and Technology**
Date of Defence:	**June, 2006**
Degree-Conferring-Institution:	**Harbin Institute of Technology**

摘要

随着Web信息的急速膨胀，各项和Web有关的服务也都逐渐增多，Web信息在很多方面得到了广泛的应用，人们对于Web信息的要求也越来越高，使得专门负责Web信息采集的网络爬虫技术面临了一个巨大的挑战。国内外的一些大公司对这一问题已经有了很成熟的解决方案，并已投入使用，但是这些大型搜索引擎只能给大众用户提供一种普通的不可制定的搜索服务，它不可能考虑到所有用户的各式各样的需求，而单机的网络爬虫在很多情况下又难当重任，中型规模的网络爬虫以其灵活的可定制性和单机网络爬虫无法比拟的信息采集速度和规模，满足了人们日益增长的对Web信息的面向用户的需求，针对这一情况，本文展开了对国内外的网络爬虫技术的研究。

网络爬虫研究中最重要的是设计构架和关键技术的解决。在吸取了他人技术和经验的基础上，本文设计描述了一个分布式网络爬虫的结构设计，其中包括硬件的构架，和软件的模块划分。硬件部分由一台PC机做控制节点，N台PC机作爬行节点，在局域网中连接。软件部分又分为控制节点软件设计和爬行节点软件设计。

然后本文分析了分布式网络爬虫的关节技术的解决方法，比如分布式的各个结点如何协同工作，任务如何分配，如何保持重要网页的时新性等等，进而提出了一些实用的算法，解决了这些分布式网络爬虫的关键技术，实现了一个具备健壮性，可扩展性，可配置性的分布式网络爬虫系统，并就该分布式网络爬虫系统进行了仔细的剖析。最后在该网络爬虫上作了一些测试，包括了普通爬行测试和该网络爬虫的一个应用，网站爬行的测试。

关键词 网络爬虫；并行；搜索引擎

Abstract

With information rapidly expanding in the Web, many Web-related services came in increasing numbers accordingly. Web information is applied in many fields, and the request of people becomes more and more rigorous, so the Web crawler who has charge of gathering of Web information is facing a real challenge. A number of big companies overseas or domestic have solved it in a perfect way, and their products have been in practical use, whereas, the search engine at large-scale can only supply a common service which cannot be customized. They cannot consider the varied demands of people. Web crawler of stand-alone cannot meet the demands in most situations. The Web crawler of medium-size can be customized conveniently and has a good performance to solve the problem. Therefore, we get into the research on the technology of the distributed Web crawler.

In the research of Web crawler, the most important things are structure design and solution of the key technologies. Based on the work of other people, we described the structure design of a distribute Web crawler, which including the organization of hardware and module partition of software. In this paper, one PC is utilized as the main node, and other PCs as the common nodes which are connected in LAN. The software architecture included main node design and common node design. Then, we analyzed solutions of the major techniques of the distributed Web crawler, such as how the nodes of the crawler cooperate with each other, how the task is distributed, how to keep the important Web fresh. We have proposed some practicable arithmetic to solve the problems mentioned above. Besides, we implemented a robust, distensible, customized, distributed Web crawler, and anatomized it. At last, we gave the results of two experiments, including common test and a site download test.

Keywords Web Crawler; Parallel; Search Engine

目录

摘要.....I
Abstract.....II

第 1 章 绪论.....1
1.1 课题背景.....1
1.1.1 搜索引擎的分类和整体结构.....1
1.1.2 网络爬虫研究现状.....3
1.2 工作意义及论文工作.....4
第 2 章 分布式网络爬虫基本构架.....6
2.1 设计目标.....6
2.2 分布式网络爬虫结构设计.....7
2.2.1 爬行节点的结构设计.....7
2.2.2 控制节点的结构设计.....14
2.3 本章小结.....14
第 3 章 分布式网络爬虫的关键技术.....16
3.1 种子集合的选取.....16
3.2 分布式策略.....16
3.2.1 分配策略分类.....16
3.2.2 分配策略比较.....17
3.2.3 任务分配粒度大小的选择.....18
3.2.4 分配函数的选择.....19
3.2.5 作为网站下载的网络爬虫的任务分配实现.....22
3.3 多线程下载.....24
3.3.1 多线程介绍.....24
3.3.2 多线程带来的问题及解决方法.....26
3.4 网页分析.....28
3.4.1 HTML中的标记.....28
3.4.2 页面链接的提取.....30
3.5 网页更新.....31
3.6 本章小结.....33

第 4 章 系统实现及实验评测 34
4.1 系统实现 34
4.1.1 分布式任务分配的实现 35
4.1.2 单结点下载任务的实现 37
4.2 系统评测 39
4.2.1 普通爬行评测 39
4.2.2 网站全站爬行评测 41
4.2.3 本章小结 43
结论 44
参考文献 45
哈尔滨工业大学硕士学位论文原创性声明 48
哈尔滨工业大学硕士学位论文使用授权书 48
哈尔滨工业大学硕士学位涉密论文管理 48
致谢 49

第1章 绪论

1.1 课题背景

随着国际互联网（internet）的迅速发展，网上的信息越来越多，全球目前的网页超过 20 亿，每天新增加 730 万网页。要在如此浩瀚的信息海洋里寻找信息，就像“大海捞针”一样困难。搜索引擎正是为了解决这个问题而出现的技术。搜索引擎是通过互联网搜索信息的重要途径。它要用到信息检索、人工智能、计算机网络、分布式处理、数据库、数据挖掘、数字图书馆、自然语言处理等多领域的理论和技术，具有很高的综合性和很强的挑战性。本文研究的内容是作为搜索引擎关键的一部分的网络爬虫，首先，简略介绍一下搜索引擎。

1.1.1 搜索引擎的分类和整体结构

搜索引擎虽然所采用的技术和实现的方法各有不同，但是总体来说可以分为两类，一种是基于目录的搜索引擎，另一种是基于全文检索的搜索引擎。

早期的搜索引擎很多都是基于目录的搜索引攀，由于人工干预较多，故在覆盖的范围上要远远的小于基于信息采集器的搜索引擎。一般来说，由于使用了人(专家)来对网站进行归纳和分类，网站分类技术为网络信息导航带来了极大的方便，广受人们的欢迎。但是它的缺陷除了成本较高之外，对网站的描述也十分简略，其描述能力不能深入网站的内部细节，因此用户查询不到网站内部的重要信息，造成了信息丢失[1]。例如：如果对一个进行电脑硬件销售站点的目录分为是商业与经济>公司>电脑硬件公司。对其描述为“显示器、电源、硬盘内存等销售”。用户在以“显示器”、“硬盘”为关键字进行检索时，就能检索到。但如果该站点中还包含有对于主板和显卡的介绍，用户在检索“主板”、“显卡”时，就无法检索到了。同时，对于基于目录的搜索引擎技术而言，其覆盖范围相对与基于全文检索的搜索引擎而言十分有限。

目前，在国内外各主要商业搜索引擎在技术上主要使用了全文检索技术，下面对基于使用全文检索技术的搜索引擎进行讨论。

基于全文检索技术的搜索引擎主要由三部分组成，如图1-1所示，信息采集器（网络爬虫），索引器、搜索接口。

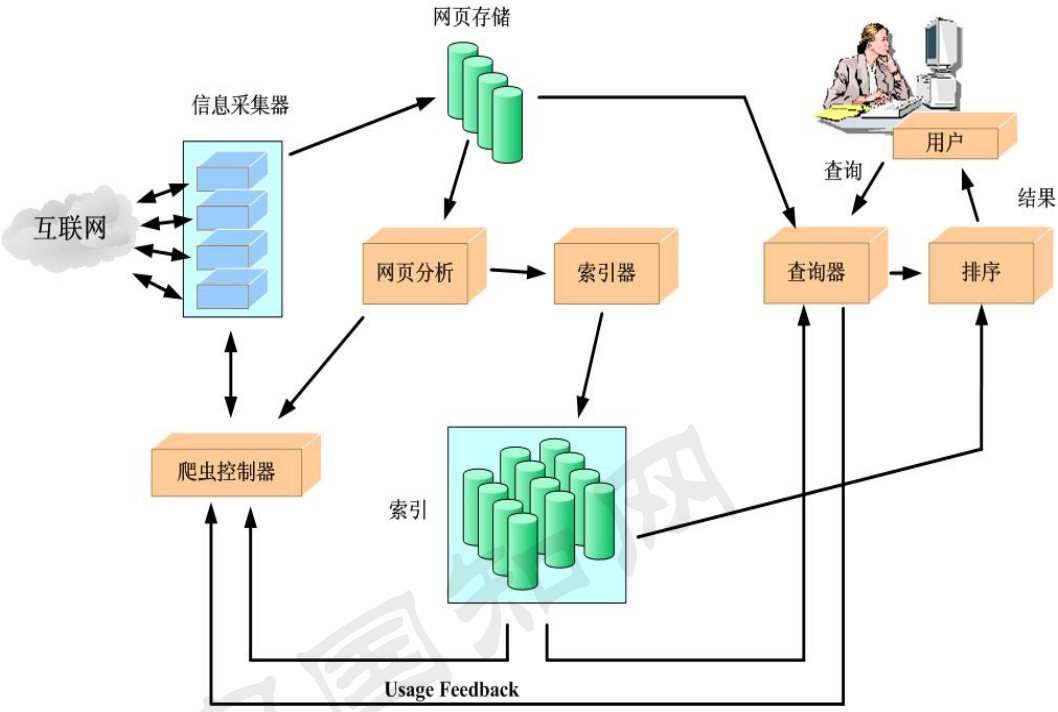

图 1-1 搜索引擎的基本构成

Figure 1-1 Frame sketch map of search engine

信息采集器：主要功能就是搜集互联网上的信息资源（主要是网页和文字信息资源）。运行信息采集器时，只要提供极少量的起始网页，信息采集器就能够按一定的规则沿着网页上的超级链接在网络上漫游，收集资源信息，直至遍历整个网站。它的性能在很大程度上影响了搜索引擎站点的规模。这部分是本论文要讨论的重点。

索引器：由信息采集器从网上取来的信息杂乱无章，如果把它们直接用于查询，效率将极为低微。索引器的主要功能就是分析收集的信息，建立索引库以供查询。它主要用到的技术有分词、索引词选取、停用词过滤、索引归并、索引压缩、索引更新、倒排文件缓存。

查询接口：它是用户与搜索引擎的接口。它通常是一个Web应用程序，主要负责接收、解释用户的请求、查询索引库以及返回排序后的搜索结果。它的用户界面友好与否是用户能否最大限度地使用搜索引擎功能的关键。

信息采集模块主动派出信息采集器进行自动搜索，信息采集器自动地在网上漫游，从一个URL或一组URL开始，访问该URL，记录该URL所指文件中所有新的URL。然后再以这些新URL的为起点，继续进行本地索引，直到再没有满足条件的新URL为止。对于一些新出现的网站或在自动搜索中有所遗漏的站

点，用户也可以自行向搜索引擎提交网站地址，使得站点内容能被及时得以搜索。得到网页内容后，信息预处理模块过滤文件系统信息，为文件系统的表达提供各种满意的索引输出，获取最优的索引记录，使用户能很容易地检索到所需信息。信息预处理模块要完成以下一些功能：格式过滤、词语切分、词性标注和短语识别等。

最后这些被处理完的信息被送入一个数据库中，使用者在执行查询时，实际上是从这一数据库中寻找匹配网页、或资料的过程。

全文检索已是一个比较成熟的技术，它能够解决对大量网页细节的检索问题。从理论上说，只要网页上出现了某个关键词（如果是中文，这个关键词必须是一个词或者是词的组合），就能够使用全文检索用关键词匹配把该网页查出来。

1.1.2 网络爬虫研究现状

网络爬虫，又称为Robots，Spiders以及Wanderers，几乎与网络同时出现。第一个网络爬虫是Matthew Gray的Wanderer，出现于 1993 的春天。在头两届国际万维网会议上出现过数篇关于网络爬虫的论文，如文献[2~4]。但是那时候互联网上的信息规模比现在要小得多，那些文章中并没有阐述如何处理现在所面临的海量网络信息的技术。每个搜索引擎的后台，都有相应的网络爬虫在工作着。但是出于互相竞争的原因，这些网络爬虫的设计并没有公开，除了以下 3 个：Google Crawler，Internet Archive Crawler以及Mercator[5]。

搜索引擎Google中，采用了多台机器进行分布式爬行[6]。它的网络爬虫包括 5 个功能模块，分别运行在不同的进程中。一个URL Server进程负责从一个文件里读取URL（Uniform Resource Locator），并把它们分发给多个Crawler进程。每个Crawler进程运行在不同的机器上，采用单线程和异步I/O同时从近 300 个网站上获取数据。所有的Crawler将下载来的数据传输到同一个Store Server进程，它将这些页面压缩并存放在磁盘上。Indexer进程将这些页面从磁盘上读出，它将URL从HTML页面中抽取出来，并将它们存放在另一个磁盘文件中。一个URL Resolver进程读取这个存放链接的文件，将其中的相对链接转化为绝对链接，然后存入一个文件，这个文件供URL Server进程读取。

Internet Archive Crawler也使用多台机器进行爬行[7]。每个Crawler进程可分配 64 个站点同时爬行，并且每个站点最多只分配给一个Crawler来爬行。每个单线程的Crawler进程从磁盘中读取分配给其爬行的站点的种子URL，把它们发送到各自站点的爬行队列中。然后采用异步I/O从这些队列读取链接，下载对

应的网页。一旦一个HTML网页下载下来，Crawler就将包含在其中的链接抽取出来。如果链接指向同一个网站，那就将该链接加入到该站点的队列中；否则，就将该链接存放到磁盘中。一个批处理进程周期地将这些链接进行过滤，去除重复链接，并把它们放入相应站点的队列中。

Mercator是一个在可扩展性方面做得非常出色的Crawler[8]。Mercator完全用Java实现。它采用的数据结构可以不管爬行规模的大小，在内存中只占有限的空间。这些数据结构的大部分都在磁盘上，在内存中只存放有限的部分，伸缩性很强。Mercator采用模块化设计的思想，通过替换以及增减模块可以很方便地实现各种功能，如进行各类Web信息统计以及Web快照，体现了良好的可扩展性。Mercator由 5 个部分构成，分别负责：给即将付诸下载的URL进行排序；将主机名解析为IP地址；使用HTTP协议下载文档；从HTML文档中提取链接；检测一个URL是否已经遇到过。

一个网络爬虫程序通常网络爬虫从种子 URL 开始，通过网页内容解析，跟随网页上的超链接进行下载。互联网上的信息更新很快，必须定期更新已经搜集过的旧信息，避免无效链接，同时获取最新信息。只有高效深度的挖掘才能使搜索引擎提供全面、即时的服务。

应用于大规模系统的数据采集软件有两个主要设计要求：一是必须有合理的挖掘策略，主要是何时下载哪些页面。常用策略包括：主题式挖掘，根据 URL 链接分析和网页内容对 URL 列表进行主题分类，然后根据类别进行有目的的挖掘；分级式挖掘，根据站点规模、权威性、数据更新频率等参数将站点列表进行分级，实现等级式采集。二是必须要有高度优化的采集架构，能高速下载大量网页，要占用合理的网络流量、具有鲁棒性、易于管理。目前主要采用服务器集群技术，由中央控制软件进行任务分发、负载平衡和运行监控。

1.2 工作意义及论文工作

国内外的一些大公司对大型网络爬虫已经有了很成熟的解决方案，并已投入使用，但是这些大型搜索引擎只能给大众用户提供一种普通的不可制定的搜索服务，它不可能考虑到所有用户的各式各样的需求，而单机的网络爬虫在很多情况下又难当重任，中型规模的网络爬虫以其灵活的可定制性和单机网络爬虫无法比拟的信息采集速度和规模，满足了人们日益增长的对 Web 信息的面向用户的需求，因此本文着眼于中等规模，力求实现一个健壮性，可扩展性，效率各方面都很完善的一个高质量的爬虫。在这个爬虫的基础上可以开展很多

研究比如：普通搜索引擎，主题搜索，移动搜索等等。

本论文的组织结构如下：

第一章绪论，介绍了课题的背景。

第二章分布式网络爬虫基本构架，介绍了分布式网络爬虫的设计目标，以及分布式网络爬虫的结构设计。

第三章分布式网络爬虫的关键技术，介绍了分布式网络爬虫在具体设计的时候会遇到的一些难题，讨论并得到解决的方案。

第四章系统实现及实验评测，介绍了系统实现的环境和具体的实现方法，并对本文介绍的分布式网络爬虫系统进行了实验评测。

第2章 分布式网络爬虫基本构架

本章讨论一个良好的网络爬虫的设计目标，然后讨论单个结点的结构设计，最后讨论分布式网络爬虫的结构设计。

2.1 设计目标

现在讨论一个网络爬虫都有哪些要求，以及如何实现这些要求，具体的细节将在后来的几章中详细介绍。

1、灵活性 在很多种不同的应用中都可能会用到这个网络爬虫，所以需要设计成为一个有足够的灵活性的网络爬虫，以至于在不同的场合只需要做出很少的改动就能应用它。

2、低成本和高性能 系统应该能够处理大规模的网页，并且运行在一个低成本的硬件环境中，本文设计该系统运行在不同的 PC 集群上，由于硬盘的读写效率太低，所以如果要维护一些主要的数据结构，还是需要比较大的内存

3、健壮性 它包括了几个方面。第一，由于系统需要和成千上万的服务器通信，所以它必须能够有处理不能连接 URL 的这种情况，异常的服务器反应等情况。对于这些情况一般设定一定的反应时间，如果超过了预定的通信时间，则判定该 URL 不能够下载，就跳过，同样，如果服务器出现异常举动，就跳过整个服务器的 URL。第二，本文的系统是有多个 PC 机组成的，在长时间的运行后，每个 PC 机结点都可能出现软件或者硬件上的错误，本文的任务切分算法必须考虑到这些情况，并且需要有监测模块监视是否有 PC 机结点出现故障，并启动错误处理程序。第三，一个网络爬虫的一次爬行是需要数周甚至数月运行的，就要求系统能够在系统出现致命故障，停机后，或者是网络阻塞后能够保护好现场，以便恢复后能够断点续穿，不需要随时对系统的数据进行备份，只需要对其进行阶段性的备份即可，这样就能在出故障后，只用重新下载少量的网页就可以继续故障以前的爬行。

4、速度控制 严格的遵循爬行规则是很重要的，它包括robots.txt和robots元标记[9]。还需要用不同的方法控制下载的速度。必须防止在同一时刻将过多的网络负载放在一个服务器上面，要规定以下访问的间隔，最好是对同一个服务器每隔 30s访问一次。

5、可操作可配制性 系统需要一个运行的时候的对外接口，用来监视爬虫

的一举一动，包括爬行的速度，web 服务器的个数和网页数统计，和主要的数据结构的大小。管理员应该能够调整爬行的速度，添加删除组件，停止爬行，恢复爬行，将某些 web 服务器或是域名加入到一个类似 blacklist 的列表中去。在一次重大的故障发生后，可能要重新配置机器，以及软件的参数，所以在爬虫运行一段时间后，它的样子会和一开始的时候大不一样的。

分析完一个网络爬虫系统的设计目标后，再来讨论一下分布式网络爬虫的设计结构，这个系统是由多台 PC 集群组成的集中式的网络爬虫。

2.2 分布式网络爬虫结构设计

如图 2-1 所示分布式网络爬虫由多台 PC 机组成，其中一台是控制节点，其他为爬行节点，控制节点负责维护所有结点的信息，对所有结点的信息进行同步，同时在控制节点上进行结点的添加和删除。

2.2.1 爬行节点的结构设计

单个结点的设计结构如图 2-2 所示，可以分为五个模块：

1、URL 分配模块

2、结点通信模块

3、URL 分析模块

4、下载模块

5、网页分析模块

2.2.1.1 URL 分配模块

本模块主要的任务是：协调各个结点的工作，将任务分配给不同的结点，保证各个结点的工作没有重复，并且能够添加，删除结点。

2.2.1.2 结点通信模块

结点通信模块负责结点间的相互通信，除了采集器采集网页时直接与 Internet 交互外，其它时候所有网络通讯都通过通信器完成。这样的设计，有两个好处：第一，其它模块只需关注本身的策略，而不用关心具体的通讯细节；第二，通讯模块与其它模块松耦合，系统可以构建在不同的对等网络上，只要通信器运行不同的协议即可。

Send (node_num, Packet)：上层模块即是通过调用 Send，把它的数据 Packet 以 node_num 为目标标识发布。接收到来自上层的发送请求后，通信器首先将 Packet 缓存起来。它再以 node_num 为目标标识通过每个结点上维护

的结点号和 IP 对应的结点信息列表获得目标结点的 IP 地址，接着再将 Packet 直接传输。

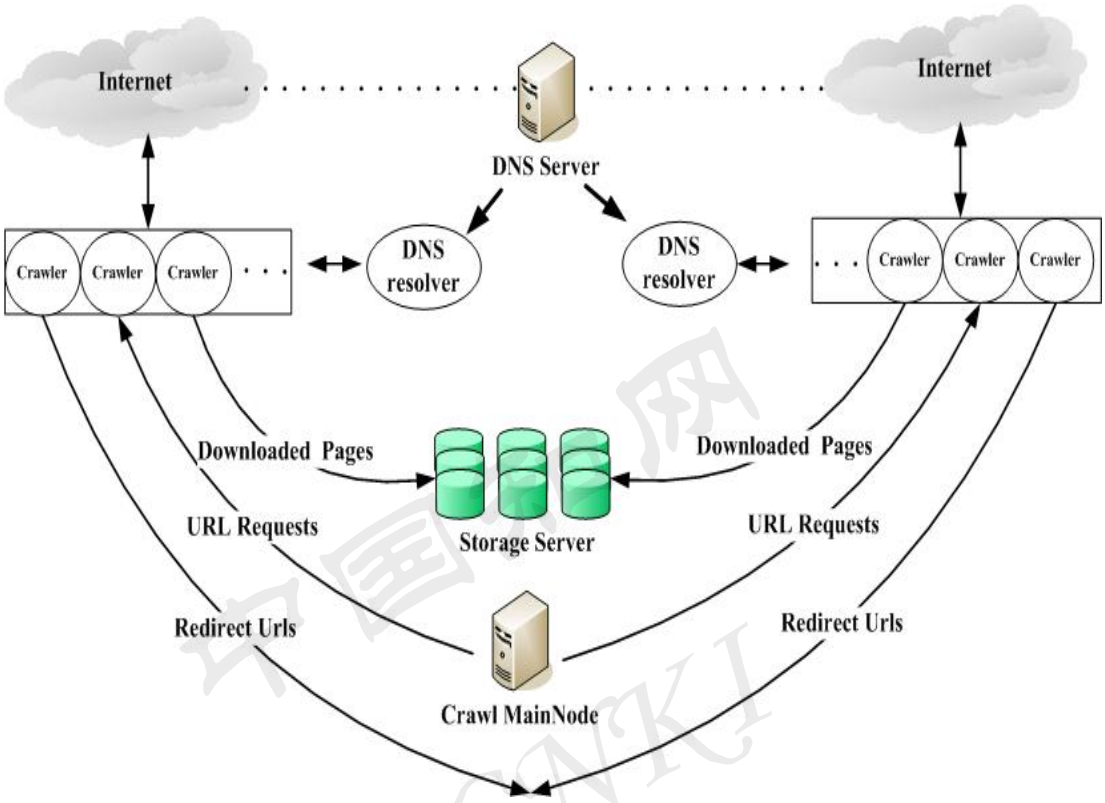

图 2-1 分布式网络爬虫结构设计图

Figure 2-1 Distribute web crawler frame design

Recv(Packet)： 该调用由上层各个模块提供，是个回调操作。结点通信模块从网络收到别的结点发送的 Packet 时，首先判断该 Packet 具体发往哪个模块，确定后它将调用相应模块提供的 Recv 操作。

2.2.1.3 URL 分析模块

URL 分析模块的任务是接受来自分布式模块分配的 URL，判断该 URL 是否访问过。若访问过，则忽略，否则加入即将访问 URL 列表。

本模块由三个子模块构成（如图 2-3）：

1、维护即将访问 URL 队列模块

2、维护已访问 URL 队列模块

3、IP 与域名转换模块

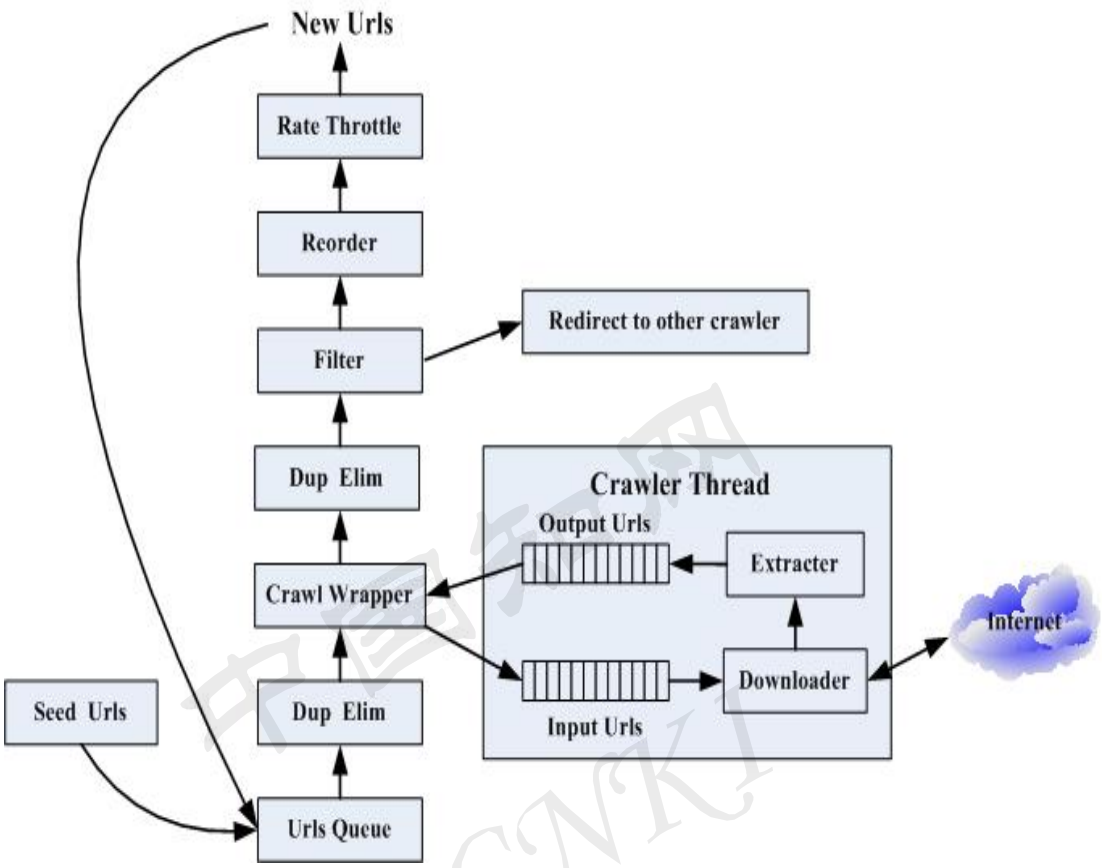

图 2-2 爬行结点结构图

Figure 2-2 Crawler node structure design

维护即将访问URL队列模块：在每个结点中维护一个本结点将要访问的URL对列模块。本文参照Mercator系统的经验将该数据结构设计成为并行的多道FIFO队列[10]，每一道指向一个domain，本结点上不同的线程在不同的队列中取URL，这样就保证了不会有多个线程同时访问一个domain。

由于这个多道队列的数据结构在爬行进行到一定阶段的时候会十分庞大，内存不能够承载，这样必须将队列的中间部分放在硬盘上，在内存中只保留队列的头部和尾部 。

维护已访问 URL 队列模块：该模块定义了两个操作：Insert(URL)，Contains(URL)。Insert 是将 URL 插入到队列中的操作，Contains 是给出一个 URL 返回一个 Boolean 显示该 URL 是否在队列中出现。

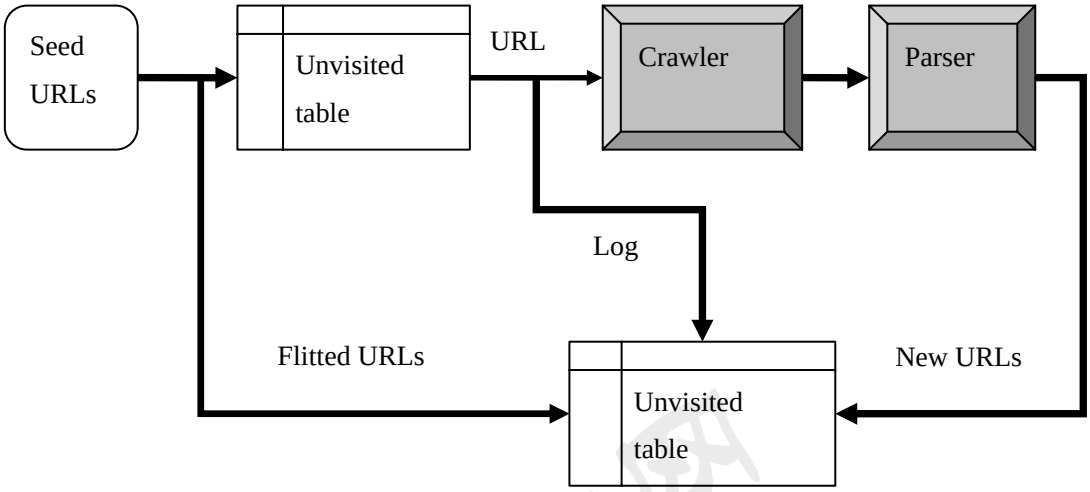

图 2-3 URL 分析示意如图

Figure 2-3 URL analyze sketch map

完整的 URL 是没有必要记录的，只需要判断 URL 是否已下载即可，因此本文用一个 hash_table 存储 URL 的 checksums，高位储存 URL 的 hostname 的 checksum，这样来自同样 domain 的 URL 在表中会排列在一起。同样由于爬行一段时间后这个表会很庞大，只能将它放在硬盘上，但是会给它建立一个 LRU cache，由于网页链接的聚簇性和实现定义的高位存储 hostname 的数据结构，使得硬盘的存取以较小的概率发生，提高了系统的速度。

IP 与域名转换模块：该模块主要解决 URL 不同但指向同一物理网页带来的问题。只要域名与 IP 不是一一对应的都会造成重复收集网页，造成这种情况有以下三种原因：

1、虚拟主机

虚拟主机就是指多个域名共同拥有一个 IP，但是各域名下的内容不同，即各个站点下的内容不同，可以认为是多个 web 服务器。

2、DNS 轮转

这种情况主要出现在商业站点中，因为商业站点的单位时间的访问量比较大，需要复制不务器的内容，通过 DNS 轮转才能达到负载平衡，减小用户的响应时间，满足用户的需要[11]。新浪网站的访问量就很大，为了满足用户的需求，他们就使用了 DNS 轮转的方法，使得在不同的时间内，同一域名对应的 IP 是不一样的。

3、一个站点多个域名

这种情况就是多个域名，其实对应的是物理上唯一的一个站点。

本模块实现方法：不能简单地根据 IP 地址来判断是否为同一个站点，因为有虚拟主机的情况。要找出那些指向统一物理位置 URL 的多个域名和 IP。这是一个逐渐积累的过程。首先要积累到一定数量的域名和 IP，下载一些网页判断分别属于那种情况，分别记录下来，供以后避免重复收集使用。

2.2.1.4 下载模块

下载模块的任务是由上级模块提供 URL，下载该 URL 所指网页。

本模块有三个子模块：

1、下载线程

2、结点线程控制

3、DNS 模块

下载线程主要解决本地结点和 web 服务器的通信问题。

在HTTP1.0 中即使客户希望在同一次会话中从服务器端传输更多的HTML页面，该TCP也要终止，而在HTTP1.1 中为了节省时间和网络带宽，可以保留上次已经建立的连接，如果该连接没有失效，本次继续使用[12,13]。

如图 2-4 所示，服务端接受搜集端发送的请求消息后，先返回一个 HTTP 头信息（为方便起见，后面称“网页头信息”），其中包含文件类型，大小，最后修改时间等内容；接着是两个“\r\n”，表现为一个空行；然后返回 HTTP 体信息，其中包含网页的全文内容。

根据网页体的大小，申请内存空间准备接收，如果超出预定接收大小，放弃该网页；根据网页类型，判断是否获取该网页。如果满足获取条件，继续进行网页体信息的获取。

注意网页头信息中给定的网页体大小有可能错误，所以读取网页体信息应该是处在一个循环体中，直到不能读到新的字节为止。因此根据需求增减内存空间，是格外需要注意的地方。在读取网页信息时，存在服务器长时间不响应的情况，为了加快搜集效率，要设定超时机制，超时后放弃该网页。接收的数据如果超出预定接收大小，放弃该网页。

结点线程控制子模块根据本结点的计算机硬件的运行情况，把并行开设的线程数控制在一个最佳的数量上，并监控保证同时访问同一个domain的线程数不超过n（通常为 1 或者 2），这样才能保证web服务器不会出现类似于拒绝服务攻击(DOS)的反应而使得一些页面的漏取。最基本的DOS攻击就是利用合理的服务请求来占用过多的服务资源，致使服务超载，无法响应其它的请求[14,15]。Web服务器的管理人员为了防止这种恶意或者非故意，对服务器造成

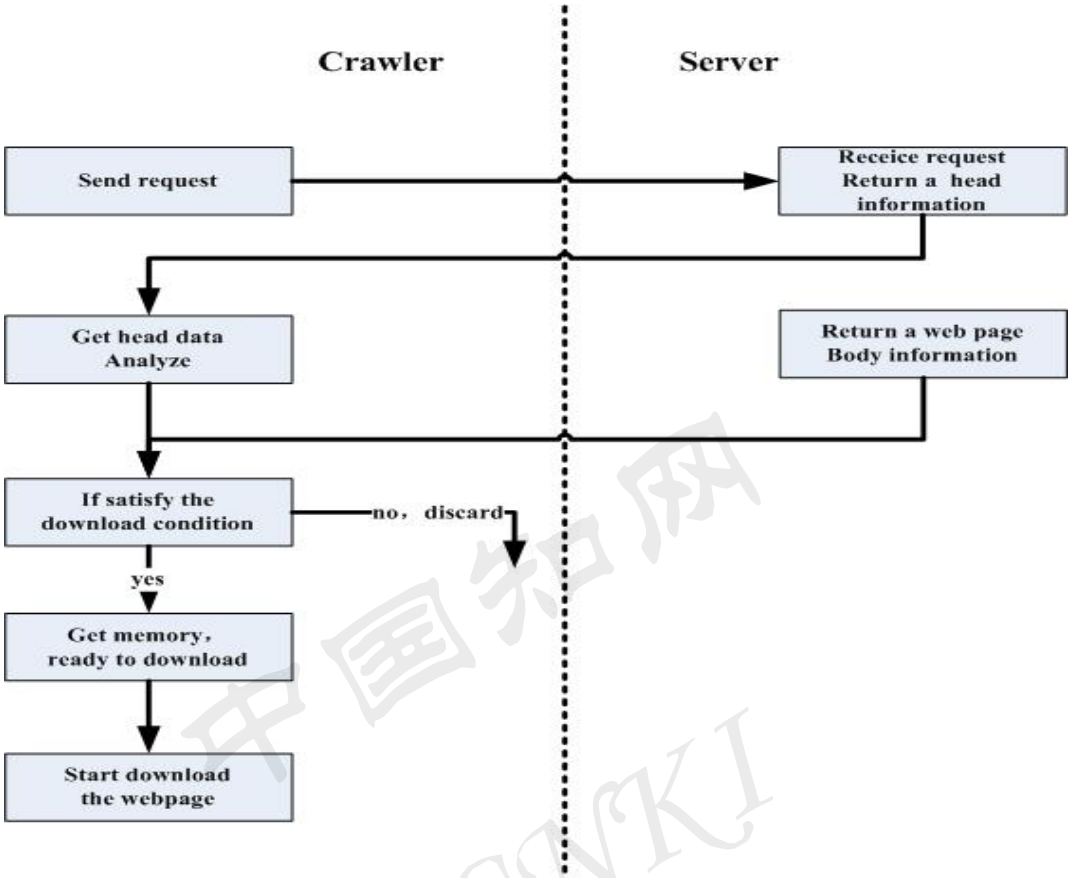

图 2-4 系统与 web 服务器的会话

Figure 2-4 Conversation with server

故障的行为做了一些防范措施，比如从同一个 IP 发出的 TCP/IP 请求必须小于一定的数目，否则视为攻击，所以本文的系统在和 web 服务器通信的时候必须避免这种情况的出现。

DNS 模块：该模块建立了一个 DNS 缓存，如图 2-5 所示。如果没有 DNS 缓存，每次搜集新的 URL 时，即使在上一次的任务中刚刚解析过该主机名，也要重新进行域名解析，所以为了避免频繁的查询 DNS 服务器需要建立 DNS 缓存。

2.2.1.5 网页分析模块

网页分析模块的任务是从网页中提取需要的 URL，如图 2-6 所示。

本模块实现起来比较简单，但是必须做到有很快的速度和提取尽可能多的 URL。

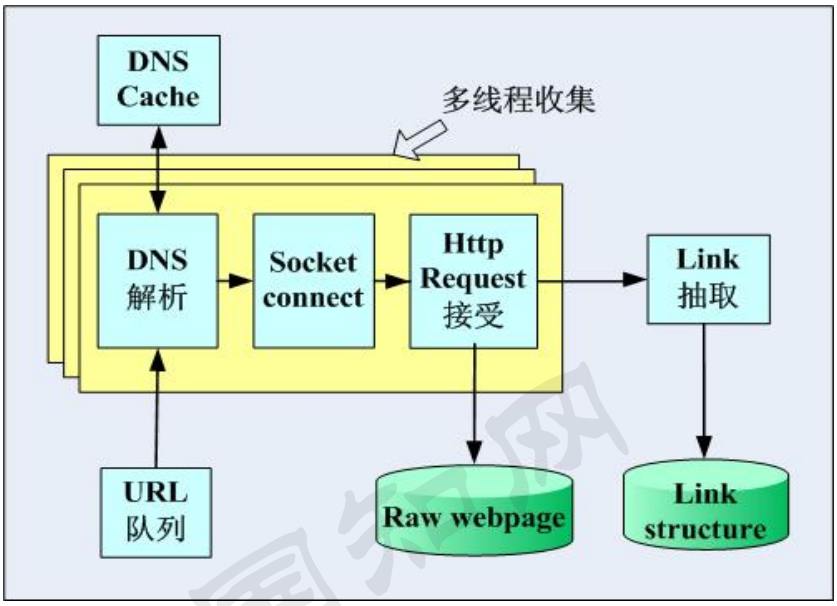

图 2-5 DNS 示意图

Figure 2-5 DNS sketch map

由于 HTML 的灵活性，URL 出现的语境有很多种情况，所以首先要参照 HTML 的语法，给出相应的 URL 出现的语境。

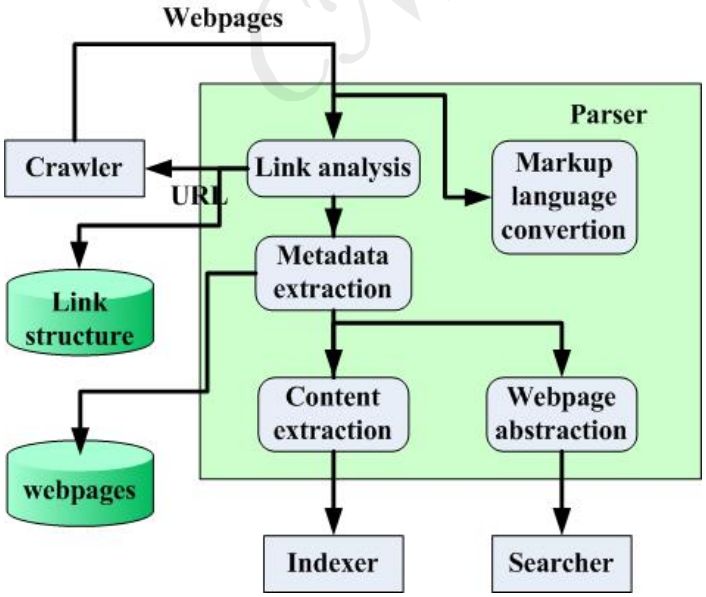

图 2-6 网页分析示意图

Figure 2-6 Web parser sketch map

2.2.2 控制节点的结构设计

控制节点主要由有三个部分组成：

1、运行状态观察及参数调整

2、添加删除结点

3、结点有效性监控

2.2.2.1 运行状态观察及参数调整

首先要能够在控制节点观察到各个结点运行的状态，比如各个结点的运行时间，总共下载的网页数量，分析的 URL 数目，最近运行情况的直方图，等等。根据这些数据用户能够更好的调整参数使得整个系统能够在最佳的情况下运行。

参数调整是很有必要的，在文献[16,17]中提到很多 crawler 在运行很长一段时间后，由于参数的不断调整，和最初的样子会大不一样，由此也能看出参数调整的重要性，下面列举一些比较重要的参数：每个结点的并行下载的线程数目，DNS 连接数，一个网站爬行的深度，静态网页的爬行深度，动态网页的爬行深度，同一个服务器的访问间隔，下载文件的后缀名字，还有一些缓冲区的大小等等，众多的参数使得系统有很好的可配制性，使系统能够适应不同运行的环境，在不断的调整后，系统能都达到性能的最佳。

2.2.2.2 添加删除结点

添加删除结点是两个很有必要的操作，正是这两个操作的存在使得系统有了可扩展性和容错性，健壮性。实现这两个操作的函数是：Add_node()，Delete_node()。Add_node()在结点信息表中添加一条记录，Delete_node()则在结点信息表中删除一条记录，同时触发结点信息同步操作使得每个结点上存在的这个结点信息表的副本更新为当前的最新表。

2.2.2.3 结点有效性监控

结点有效性监控模块，是整个系统唯一的一个同步模块，它利用 timeouts 来检测是否有 crashed 结点，如果有，则在结点列表里将该结点删除，同时在各个结点中的 hash 映射表里将该结点的信息删除，防止给该结点分配新的任务。

2.3 本章小结

本章主要介绍了分布式网络爬虫的设计目标和分布式网络爬虫的结构设计

方案。其中结构设计方案又分为：爬行节点的结构设计和控制节点的结构设计。

第3章 分布式网络爬虫的关键技术

3.1 种子集合的选取

如果种子集合的选择不好可能会使得搜索到的网页只占总网页数很少的一部分。

一种方法是在第一次人工干预的全面网页搜集后，系统维护相应的URL集合，往后的搜索或者基于这个集合或者通过某种策略在这个集合中选择具有代表性的URL作为下一次搜索的初始种子集合。另一种方法是让网站拥有者主动向搜索引擎提交他们的网址（为了宣传自己通常有这种积极性），系统在一定时间内定向向那些网站派出爬虫程序[18,19,20]。

3.2 分布式策略

3.2.1 分配策略分类

按照地域来划分，分布式信息采集分为两类：基于局域网的分布式信息采集和基于广域网的分布式信息采集[21,22]。前者的所有采集器都运行在同一个局域网内，它们之间通过高速的内连接进行通信，完成采集；而后者的采集器全部被分布在地域上较远的Internet上，通过网络进行远程通信。在实际应用中，基于局域网的分布式信息采集应用较广，这是因为在个性化信息采集发展的今天，大多数采集目标并不是整个Web，它们的采集范围要小的多；另外，基于广域网的信息采集系统的实现要复杂的多；还有，基于广域网的信息采集的设计和运行成本也较昂贵。然而，对于资金实力雄厚和采集任务较重的大公司，特别是采集目标面向整个Web的门户搜索引擎，多采用广域网模式，或者广域网和局域网的结合模式[23]。

整个Web可视为一个巨大的有向图[24]，其中的结点就是实际中的网页，而结点间的有向边则代表网页间的超链接。由于下面讨论的是把Web 采集任务进行划分后，各个任务的分配模式，并不涉及对链接的处理过程，因此可以把Web 有向图简化为一个所有网页的集合G。任务分配的目的是使得各采集进程间达到负载均衡，在不考虑采集机器到各Web 站点的通信带宽及各Web 服务

器的性能和负载差异的情况下，各采集进程的任务量可以简单地用下载的Web页面数来衡量。如何来对整个集合G 进行完全分配，是各种任务分配模式要研究的问题。

设 URLs= {URL1, URL2, …}是所要完成收集的网页地址集合。这是一个开放和动态变化的集合。所谓“开放”，指其中元素的个数是事先未知的，具体有哪些元素当然也事先未知。所谓“动态变化”，指它在收集过程中随着新发现的地址增加而变化。通常，一次搜集过程由某些“种子”网页开始，沿着它们包含的超链，按照某种搜索策略（先宽，先深，等等）往下进行，直到没有新的地址发现，或者人为决定不要再进行了（例如磁盘已满）。

按照各个结点的协作方式，分布式网络爬虫可以分为三类：

1、独立方式：各个结点之间独立的采集各自的页面，互不通信；

2、动态分配方式：对于 N 个采集结点，按照一定规则把 Web 页面集合 G 划分为 M (M > N) 个任务子集 G1 ，G2 ， … ， GM ，并且满足静态分配模式中的子集划分的前两个条件，但不必满足第三个条件。 然后把前 N 个子集的任务分配到 N 个采集结点进行处理，每当一个采集进程完成任务后，系统把下一个待分配的任务子集分配给该结点进行处理， 各采集结点如此一直进行下去， 直到所有子集都被分配完[25]。

3、静态分配方式：将 URL 按事先划分，分配给各个采集结点。对于静态分配方式，存在一种跨区链接（指一个采集结点在提取链接时遇到的不属于自己采集范围的链接）。难题在于跨区链接并不一定能被它所属于的采集结点发现，如果本采集结点采集了则可能重复采集，如果不采集则可能漏采。近期的研究工作针对这种情况比较了三种模式：丢弃模式：完全不采集不属于本结点的页面；交叉模式：采集遇到的交叉页面；交换模式（当采集到属于别的结点的 URL，就将这些链接保存起来，积累到一定数量后传输给它们所属于的采集结点进行采集[26]。

3.2.2 分配策略比较

独立方式下各个结点不能够协同工作是不可取的。而在动态分配模式下，任务子集的划分相对简单，易于管理和维护，并且能够保证所有的采集结点都处于忙的状态，充分利用了系统资源，考虑到了各采集结点间性能的差异，但是每个结点与控制节点的通信量会大大增加，结点间的通信负载过大。

下面讨论静态模式下的任务分配：设目前有 N 个采集结点，静态分配是

先把 G 划分为 N 个子集：$G_1, G_2, ..., GN$, 并且使得： $G = G_1 \cup G_2 \cup ... \cup GN$；

$G_1 \cap G_2 \cap ... \cap GN = \Phi$； $|G_1| \approx |G_2| \approx ... \approx |GN|$。然后把各个子集的采集任务分配到对应的采集结点，每个结点负责各自的采集区域，共同完成对整个 Web 的遍历采集。同时由于各子集的页面数大致相当，因此达到任务量均衡的效果。但对 Web 页面集合通常不能进行任意的划分，而是遵循一定的规则，否则会造成划分管理和维护上的极大开销。如上面所介绍的，对任务的划分通常有基于 URL 的哈希值、基于站点名的哈希值和基于域名等的划分方式。其中基于站点名哈希值的任务划分方式可以达到较好的任务量均衡效果，当哈希值的任务划分区间选择适当，各区间内的站点数相当大时，每个区间内各站点的页面数目分布情况大致相同，也就是说不同区间内站点的平均页面数趋于相同，从而达到任务量的均衡效果。下面具体介绍任务的划分。

让 n 表示并行收集系统的结点数，Sn = {0,1,2,…,n-1}表示网页搜集结点的集合。经验告诉我们，每个结点启动 200 个左右并发抓取进程是比较合适的，它们共享（争用）CPU、网卡和硬盘资源。从原理上讲，可以考虑从 URLs 到 200×n 个进程之间的映射。但本文认为所带来的管理复杂性要比可能带来的好处大得多。于是本文考虑粗一些的粒度，将目标设定在 URLs 上形成一个“优化的”n-划分。而在结点内部的并发进程则以一种自由的方式来共同完成分到该结点的任务。

优化就是要在负载平衡和降低开销之间求得一种最好的折中。由于每个任务的执行时间可能相差很大（受网页的大小、网络状态的波动等的影响），简单的按照任务个数的平均分配即使对于负载平衡也是没有意义的。 因此要慎重的选择任务分配中粒度的大小。

3.2.3 任务分配粒度大小的选择

通常任务分配的粒度大小有以下几种：

1、随机划分

这种方法简单，对于任何网页只要随机判断采集还是不采集。它有两大缺点，首先是可能导致各个结点采集的重复度较高，不利于提高整个系统的网页覆盖率；其次，当采集器需要重新采集一些网页时，可能产生不同的划分结果，将不利于分布式计算文档重要度。

2、根据网页

即通过散列函数对网页 URL 进行散列，根据散列值决定网页由哪个采集器负责并将 URL 发往相应的结点。它的缺点是需要在结点间传递较多的URL，通信开销较大。

3、根据网站

即散列的对象为网页 URL 中的网站部分。而结果表明网页中的 90%的链接都指向本身的站点，这意味着它将大大降低通信的开销。HOST(URL)，一个网页地址的域名部分（或称主机部分），通常对应某台 Web 服务器。于是可以看到在 URL 上有一个按照域名的自然划分（即 URL1 和 URL2 同属这个划分的一个“块”，当且仅当 HOST(URL1)=HOST(URL2)，记为 HOST(URLs)，即它的每一个元素是一台主机上所有网页的集合。在不至于引起混淆的情况下，也可以方便地将 HOST(URLs)的元素看成是所涉及主机的域名。

本文采用的是第三种方法，基本思路是对 HOST(URLs)进行随机分配，即建立一个从 HOST(URLs)到 Sn 之间的映射。一旦一个 HOST(URL)映射到了某一搜集结点，该搜集结点就要负责 HOST(URL)下面所有网页的收集。尽管不同的 Web 站点含有的网页数量会相差很大，但由于 HOST(URLs) ≫n，并且如果分配函数足够随机，则可以认为分到各个结点上的网页个数比较均匀的可能性很大。

3.2.4 分配函数的选择

分配函数就是指如何计算每个新出现的 URL 应该由哪个结点负责，也就是说分配函数的选取就是 hash 函数的选取。

本文分配任务的颗粒度已经讨论过，选择了根据网站及 HOST（URL）。所以最容易想到的一个分配函数就是：

$$node\_num = hash(new\_url.host)\%node\_sum\_num \quad (3\text{-}1)$$

其中 node_num 是这个 URL 将发往的结点号，new_url.host 是新解析出来的 URL 的 host，node_sum_num 是结点的总数。

上述方法是一种静态的分配算法，不利于系统的容错性和扩充性。一个好的hash函数应当满足以下几个条件[27]：

1. 平衡性（Balance）

平衡性是指哈希的结果能够尽可能分布到所有的缓冲中去，这样可以使得所有的缓冲空间都得到利用。很多哈希算法都能够满足这一条件。

2. 单调性（Monotonicity）

单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到新的缓冲中去，而不会被映射到旧的缓冲集合中的其它缓冲区。

简单的哈希算法往往不能满足单调性的要求，如最简单的线性哈希：

$$X \to ax+b \bmod(p) \qquad (3\text{-}2)$$

在上式中，P 表示全部缓冲的大小。不难看出，当缓冲大小发生变化时（从 P_1 到 P_2），原来所有的哈希结果均会发生变化，从而不满足单调性的要求。 哈希结果的变化意味着当缓冲空间发生变化时，所有的映射关系需要在系统内全部更新。而在分布式系统内，缓冲的变化等价于 node 加入或退出系统，这一情况如果频繁发生，会带来极大的计算和传输负荷。单调性就是要求哈希算法能够避免这一情况的发生。

3. 分散性（Spread）

在分布式环境中，终端有可能看不到所有的缓冲，而是只能看到其中的一部分。当终端希望通过哈希过程将内容映射到缓冲上时，由于不同终端所见的缓冲范围有可能不同，从而导致哈希的结果不一致，最终的结果是相同的内容被不同的终端映射到不同的缓冲区中。这种情况显然是应该避免的，因为它导致相同内容被存储到不同缓冲中去，降低了系统存储的效率。分散性的定义就是上述情况发生的严重程度。好的哈希算法应能够尽量避免不一致的情况发生，也就是尽量降低分散性。

4. 负载（Load）

负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中，那么对于一个特定的缓冲区而言，也可能被不同的用户映射为不同的内容。与分散性一样，这种情况也是应当避免的，因此好的哈希算法应能够尽量降低缓冲的负荷。

按照公式(3-1)的方法，设每个结点负责的主机数为 m/n，散列函数对 n 取模，可以保证系统负载平衡。在 n→n+1 情况下，散列函数对 n+1 取模，此时可以保证系统负载平衡，但是由于模数改变使原先分配给结点 n_i 负责的 URL 可能分配给 n_j 负责，从而导致重复搜集一些已经搜集过的网页；在 n→n-1 情况下具有同样的缺陷。

本文对这种方法进行改进，采用逻辑二级映射的方法。其中一级逻辑结点可以用数组形式存储（不妨设这个数组为 A），每一个数组元素的下标即是它所代表的逻辑结点号，其中存储该逻辑结点对应的结点序号。举例说明，设 n=10 为 10 个物理结点，m=10,000 为逻辑结点的数目，系统初始：其中 A[1]到 A[10,000]称为逻辑结点。URL 经过散列函数处理后首先平均映射到逻辑结点上，再经过第二次映射将第 1 到第 1,000 的逻辑结点映射到 1 结点，第 1,001 到第 2,000 的逻辑结点映射到 2 结点，以此类推将逻辑结点平均分配到 10 个结点上。

当结点数增 1，每个结点都要让出一部分控制的逻辑结点给新的结点，即对 A 做部分修改。n_{11}是由$n_1, n_2, ..., n_{10}$匀出部分组成，其逻辑数组中对应的项被置为 11。当结点数减 1，减掉的结点要让出自己控制的主机，平均分配给其它结点，同样要修改逻辑数组中的值。

具体任务分配算法如图 3-1：

Input： 等待其他结点传来的一个 URL，或者它所管辖的抓取进程返回的一个 URL 及相应的网页

Output： 将 URL 下载或传往其他节点

1 若得到其他结点传来的一个 URL

1.1 看 URL 是否出现在 visited_table 中；

2 若得到抓取进程返回的 URL，则从 URL 对应的网页中解析出超链接 LINK。

2.1 从 unvisited_table 中分给该抓取进程一个新的 URL，并将返回 URL 放到 visited_table 中。

2.2 并对每一个新得到的 URL 的 HOST 进行模 n 散列，得到某个整数 i（0<i<逻辑结点数），在结点由每个结点上面维护的一个映射表中得到具体的结点号 num；

2.3 对每一个超链接 LINK 及其对应的整数 num

2.3.1 如果本结点编号为 num，执行 1.1 的动作

2.3.2 否则，将 link 发给结点 num

3 Return

图 3-1 任务分配算法 1

Figure 3-1 Task distribution arithmetic 1

3.2.5 作为网站下载的网络爬虫的任务分配实现

由于本文的系统面向的是中级规模的网页搜集，对于一些大型的网站如新浪，163 等的网站全站下载也是本系统的用武之地，不过由于一个网站的 host 是一样的，因此需要在前面普通任务分配基础上做出一些改动，首先任务的粒度不能再选择 host，否则同一网站的网页的哈希函数的值是唯一的，任务只能分配在一个结点上，这样是不行的，但是如果按照 URL 这个粒度进行分配的话，粒度又太小了，因此经过对各大网站的 URL 分析和总结，本文提出了一种网站全站下载多机协同任务分配算法。

下面来看一些 163 网站的网址：

http://co.163.com:80/forum/content/94_369717_1.htm

http://quote.stock.163.com:80/s/ZNH.html

http://fund.163.com:80/05/1019/08/20DPAQNV001317NC.html

http://gameweb.163.com:80/2004/zh /boss/page_3_11_2.htm

http://www8.blog.163.com:80/rss/-dkc6.xml

http://riji.163.com:80/weblog/comment/hnyzjsfm/05/2323721

http://photo.163.com:80/messages/xxo740529/

再来看一些 sina 网站的网址：

http://finance.sina.com.cn/money/lczx/20060614/03022649081.shtml

http://blog.2006.sina.com.cn/m/panyueming

http://bj.house.sina.com.cn/decor/pretywoman/index.html

http://ent.sina.com.cn/s/h/2006-06-14/11421122257.html

这些大型网站的结构如图 3-2 所示：可以看出大型网站下面都有很多子网站，而同一子网站网页的 URL 有一个共同的特点，它们的 URL 前一两个字都相同，表示了该网页是这个大网站某个子网站的网页，因此有理由猜测，这些子网站网页直接的互连关系比较紧密，而子网站与子网站网页之间的引用就没有那么的频繁，并且相差很大，经统计的确如此，于是本文就把任务分配的粒度定为子网站。

再来分析这些 URL，又发现第一个“.”号前的第一个单词很好的表示了这个网页属于子网站，因此本文的 hash 函数就可以改为：

$$node\_num = hash(new\_url\_firstword)\%node\_sum\_num \quad (3\text{-}2)$$

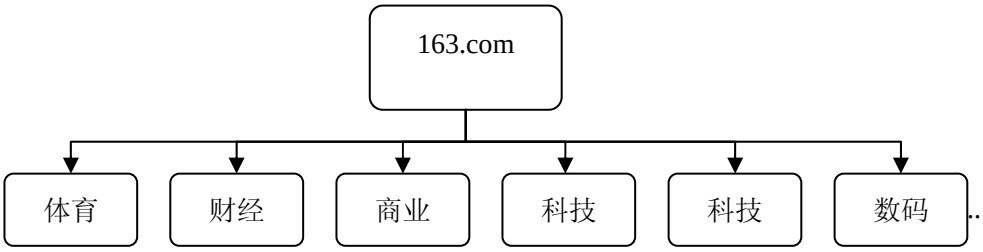

图 3-2 163 网站组织结构示意图

Figure 3-1 163 Site organization sketch map

同时,任务分配算法改变如图 3-3 所示：

Input：等待其他结点传来的一个 URL，或者它所管辖的抓取进程返回的一个 URL 及相应的网页
Output：将 URL 下载或传往其他节点
1 若得到其他结点传来的一个 URL
  1.1 看 URL 是否出现在 visited_table 中；
2 若得到抓取进程返回的 URL，则从 URL 对应的网页中解析出超链接 LINK。
  2.1 从 unvisited_table 中分给该抓取进程一个新的 URL，并将返回 URL 放到 visited_table 中
  2.2 如果爬虫用于普通爬行转 2.2.1；如果用于网站爬行转 2.2.2
    2.2.1 对每一个新得到的 URL 根据公式 3-1 计算 hash 值 i（0<i<逻辑结点数最大值），在结点由每个结点上面维护的一个映射表中得到具体的结点号 num；
    2.2.2 对每一个新得到的 URL 根据公式 3-2 计算 hash 值 i（0<i<逻辑结点数最大值），在结点由每个结点上面维护的一个映射表中得到具体的结点号 num；
  2.3 对每一个超链接 LINK 及其对应的整数 num
    2.3.1 如果本结点编号为 num，执行 1.1 的动作
    2.3.2 否则，将 link 发给结点 num
3 Return

图 3-3 任务分配算法 2

Figure 3-3 Task distribution arithmetic 2

3.3 多线程下载

3.3.1 多线程介绍

多任务操作系统运行多个单进程就可同时做多件事情；与此同时，一个进程，也可运行多个单线程，同时做多件事情。每个线程都是一个不同的控制流，可以独立地执行它的指令，允许多线程进程同时完成多项任务。一个线程在运行 GUI 时，第二个线程可以执行某些 I/O，而第三个则进行计算。多线程是一个程序在同一时刻运行超过一个任务的能力。不能把这一概念与以前的技术：多任务搞混了，后者是计算机在同一时刻运行超过一个程序的能力。两者区分是：多线程是在一个程序内部，而多任务是在一个计算机内部。在同一个时刻运行超过一个程序任务的多线程能力对 Spider 程序的效率是非常重要的。由于多线程发生在程序内部，它们使用同一内存空间，所以线程可以很容易地共享数据。

线程是一个抽象的概念，包括了计算机在执行一个传统程序时的每一件事。它是在 CPU 上进行调度的程序状态；是进行工作的“事情”。如果一个进程是由数据、代码、内核状态和一组 CPU 寄存器所组成，那么一个线程就被嵌入了这些寄存器（程序计数器、通用寄存器、堆栈指针，以及堆栈）的内容中。看上去，一个线程就是计算的状态。

多线程（MT，Multithreading）技术允许一个程序同时执行多个任务。多线程编程的基本概念存在于研究开发实验室中已有数十年之久。协同程序（Co-routine）系统（如Concurrent Pascal和InterLisp's Spaghetti stack）在 20 世纪 70 年代中期就已使用并处理了很多同类问题。Ada 任务是基于语言的结构，能直接变换到线程（当前的 Ada 编译程序可用线程实现任务）。早在 1960 年Burroughs就推出了商用带有协同程序线程的主机 OS[27] 。

编写多线程程序的主要问题是如何使各线程能协同工作。锁（同步段）和条件变量（等待/通知）是基本构件。当然并不是所有工作都需要协调，因为部分条件可以忽略。

1、内核交互作用

前面所阐明的概念和实现对所有的模型都有效。但还缺少的是线程和操作系统间关系的定义。系统如何调用工作，在 CPU 上如何调度线程，在这一级

上，各种实现的差别也很大。内核的构造不同，所提供的资源和服务也不同。

注意，在实践中 99%的线程是在这一级上实现的，而主要的差别是效率的高低。

并行性（parallelism）是两个或多个线程真正同时运行在不同的 CPU 上。在一台多处理器机上，很多不同的线程可以并行运行。当然，也是同时运行。

在多线程（MT）中，定时和同步问题的主要部分是并发性，而不是并行性。实际上，线程模型总是设计成不需用户关心并行性的细节。在单处理器（UP）上运行 MT 程序并不会简化编程问题。运行在多处理器（MP）也不会复杂化。这是一件好事。

重复这一点。如果在一台单处理器上，程序编写正确，那么在多处理器上，也会运行正确。在 UP 和 MP 上运行进入竟态条件的几率相同。如果在一个上死锁，那么在另一个上也会死锁（在几率部分有若干小的例外，但这很难出现）。然而有一些小 bug，能使程序在 UP 上运行正常，但在 MP 上出现问题。

2、系统调用

系统调用是基本功能，它可完成捕获内核中的例程。这些例程可做一些简单的事情，如为当前进程的属主查找用户 ID，也可做些复杂的事情，如重定义系统的调度算法。对多线程程序，有一个重要的问题是可能有多少线程能使系统调用同时进行。对某些操作系统，这个问题的回答是“一个”；而另一些系统，回答是“若干”。最重要的一点是系统调用同原来一样，这样全部原有程序都能运行如常，几乎没有降低。

3、信号

给信号是 UNIX 内核中断一个运行进程的方法，并让其知道发生了某种感兴趣的事情。事件可以是定时器超时或某 I/O 端口完成了一项工作或者另外的进程要进行通信。

4、同步

同步是一种方法，它可保证多线程活动步调一致。这样一个线程不能突然改变其它线程正在其上工作的数据。这是因为提供了功能调用而做到这一点，功能调用可限制能够同时访问某些数据的线程数。在最简单的情况下（一个互斥锁定－人工干预）。同时只能有一个线程执行给定的程序段，这个程序预定改变某全局数据，或完成一个设备的读或写。例如，线程T_1得到一个锁定并开

始处理某全局数据。线程T_2现在必须等待（典型的是睡眠），直到线程T_1 完成，T_2才能执行相同程序。使用同一个锁定全部（能改变数据的）程序的锁，就能保证数据的一致性。

5、调度

调度是将线程放到 CPU 上的行动（这样就可执行线程），以及将线程从 CPU 上取下的行动（这样就可执行其它线程）。实际上，调度并不经常是个问题，因为全部都按预期的那样工作。

3.3.2 多线程带来的问题及解决方法

要编写任何类型的并发程序，必须首先能可靠地同步各线程。做不到这一点会带来各种问题。没有同步，两个线程会同时开始改变数据。一个会改写另外一个。为避免这一情况，线程间必须可靠地协调其动作。

要实现同步，需要在硬件中有原子测试和置位指令。这对单处理器机和多处理器机都是如此要求。线程可在任何时间被抢先，在任何两个指令之间，都必须有一个这种指令，这样可以避免灾难性的冲击。

测试和置位指令测试（或仅加载到一个寄存器）一个来自内存的字，并将它置为某个值（典型的为 1），在一条指令内完成，没有任何可能在这之间发生什么事情（即另一个CPU的中断，或写）。如果目的字的值为 0，现在置为 1，被认为拥有锁定。如果该目的字原为 1，现在又置为 1（即没有变化），就不拥有锁定，所有同步都基于这条指令[28]。

关键段（critical section）是一段程序，这段程序要自动完成，不能有影响其完成的中断。可用锁定的办法创建关键段，处理数据，然后释放锁定。在增加计数或更新数据库中的记录时需要用关键段。与此同时，其它事情可以进行，在关键段中执行的线程甚至可失掉其处理器，但仍无其它线程进入到关键段。

如果有其它线程要执行同一关键段，需等到第一线程完成后才能执行。

关键段要尽可能地短，并要细心优化，因为它们在很大程度上能影响程序的并发性。

所有共享数据都必须用锁保护，如果没有锁定会带来问题，这一点一定要注意。传送给其它线程的数据结构以及全局变量是最明显的实例，所有能被多

线程访问的所有数据结构都包括其中，也包括静态变量。

在多线程中，等待触发是个灾难。要仔细考虑对静态的使用。如果要使用，首先要锁定它。

实现进程同步的几种方法：

互斥锁是最简单最基本的同步变量。它能提供对一个程序段的（即关键段）的单一的绝对的拥有。第一个线程锁定互斥并获得拥有，其后的任何锁定尝试都要失败，是调用线程睡眠。当拥有者解锁后，睡眠者之一被唤醒并运行，得到拥有的机会。之所以说有机会，因为其它线程会在被唤醒线程之前调用 lock 操作并拥有。这完全是正常行为，并且不应因此而影响

同样可以应用于程序同步的是信号量。在十九世纪，当火车处于发展时期，铁轨的价格高昂，因此使用单线，每次要限制火车只运行在一个方向。信号量的发明就是要让火车知道，在同一时间轨道上是否有另外的火车。信号量是垂直的杠子，上面有成 45 度或 90 度的旗帜，用以指明是否有另外的火车存在。

1960 年E.W.Dijkstra 教授将这一概念用到计算机科学。计数信号量（counting semaphore）是一个变量，它能增长到一定高度，但只能降低到 0。如果信号量大于 0，这个操作可以完成否则调用线程必须睡眠，直到另外线程增加它为止[29]。

信号量对于工作起来像火车那样的对象很有用。即所关心的为是否有 0 对象或大于 0 的对象。缓冲器和列表的填充和空白是很好的例子。当要让一个线程等待某事件时，信号量很有用。要完成这样的事情，可让线程在一个值为 0 的信号量上调用 wait 操作，当准备好让线程继续时，就让另外的线程增加这个信号量值。

条件变量也是一种实现线程同步的方法。条件是任意的，编程员负责锁定和解锁互斥、测试和修改条件以及唤醒睡眠者。此外，完全和信号量相同。

一个线程需在互斥（条件变量总有一个相关的互斥）状态下，并在互斥的保护下测试条件。不拥有互斥，其它线程不能改变条件的任何方面。如果条件为真，线程完成其任务，并在适当时释放互斥。如果条件为假，互斥释放，线程因条件变量而进入睡眠。当其它线程改变了条件时，就调用 wake 系统调用，唤醒一个睡眠的线程。用户的线程重新得到互斥，重新测试条件，或者成功，或者回去睡眠，根据条件的真假而定。必须重新测试条件！首先，其它线程在发送唤醒之前可能没有完整地测试条件。第二，即便在发送唤醒时条件为真，在线程运行前也可能改变。第三，条件变量允许虚假唤醒，它们允许没有

任何理由的唤醒。

与条件变量等价的是等待/通知（wait/notify）。行为的实质相同。进入同步段、测试条件，如果为真就继续，如果为假就等待（释放同步段）。其它线程将进入同步段，改变条件，并发送一个唤醒。重新获得同步段后，要再测试条件等。

有了以上的方法可以很容易的调整在一个结点中的线程运行数，并且使这些线程协同工作。

3.4 网页分析

3.4.1 HTML 中的标记

HTML的标记总是封装在由小于号（<）和大于号（>）构成的一对尖括号之中[30]。

1、单标记

某些标记称为“单标记”，因为它只需单独使用就能完整地表达意思，这类标记的语法是：

<标记>

最常用的单标记是<P>，它表示一个段落（Paragraph）的结束，并在段落后面加一空行。

2、双标记

另一类标记称为“双标记”，它由“始标记”和“尾标记”两部分构成，必须成对使用，其中始标记告诉 Web 浏览器从此处开始执行该标记所表示的功能，而尾标记告诉 Web 浏览器在这里结束该功能。始标记前加一个斜杠（/）即成为尾标记。这类标记的语法是：

<标记>内容</标记>

其中“内容”部分就是要被这对标记施加作用的部分。例如你想突出对某段文字的显示，就将此段文字放在一对<EM> </EM>标记中：

<EM>text to emphasize</EM>

3、标记属性

许多单标记和双标记的始标记内可以包含一些属性，其语法是：

<标记　属性 1　属性 2　属性 3　… >

各属性之间无先后次序，属性也可省略（即取默认值），例如单标记<HR>

表示在文档当前位置画一条水平线（horizontal line），一般是从窗口中当前行的最左端一直画到最右端。在 HTML3.0 中此标记允许带一些属性：

<HR SIZE=2 ALIGN=LEFT WIDTH="70%">

其中 SIZE 属性定义线的粗细，属性值取整数，缺省为 1；ALIGN 属性表示对齐方式，可取 LEFT（左对齐，缺省值），CENTER（居中），RIGHT（右对齐）；WIDTH 属性定义线的长度，可取相对值（由一对 " "号括起来的百分数，表示相对于充满整个窗口的百分比），也可取绝对值（用整数表示的屏幕像素点的个数，如 WIDTH=300），缺省值是"100%"。

4、文档结构

除了一些个别的标记外，HTML 文档的标记都可嵌套使用。通常由三对标记来构成一个 HTML 文档的骨架，它们是：

```
<HTML>
    <HEAD>
        头部信息
    </HEAD>
    <BODY>
        文档主体，正文部分
    </BODY>
</HTML>
```

其中<HTML>在最外层，表示这对标记间的内容是 HTML 文档。<HEAD>之间包括文档的头部信息，如文档总标题等，若不需头部信息则可省略此标记。还会看到一些 Homepage 省略<HTML>标记，因为.html 或.htm 文件被 Web 浏览器默认为是 HTML 文档。<BODY>标记一般不省略，表示正文内容的开始。

一个超链又称作锚（Anchor），它唯一地指向另一个 Web 信息页，超链如同把 Gopher 中可触发的菜单项融于正文之中，因此超链更具有上下文的含义。

已经知道，一个 Web 信息页是用 URL（Universal Resource Location，统一资源定位器）来唯一标识的，URL 的一般格式为：

访问方式 ： //服务器域名/路径及文件名

其中访问方式可有 HTTP、FTP、TELNET、GOPHER、WAIS、NEWS、MAILTO 等等。随着访问方式的不同，冒号后面的参数格式也会不同，下面是一些 URL 的例子：

http://www.ecnu.edu.cn/coliege/science/computer/computer.htm

ftp://ftp.sjtu.edu.cn/pub/

mailto:webmaster@www.ecnu.edu.cn

file:///c/html/sample.htm

HTML 中的一个超链由两部分组成：一部分是可被显示在 Web 浏览器中的超链文本及图像，当用户在它上面点击鼠标时，就触发了此超链；另一部分就是用以描述当超链被触发后要链结到何处的 URL 信息。因而超链标记（即锚标记）的格式为：

< A HREF="URL 信息" > 超链文本及图像 </A>

其中超链文本被浏览器用一种特殊颜色并带下划线的字体醒目地显示出来，并且用户鼠标进入其区域时会变成手的形状，表示此处可以被触发。属性 HREF 表明了超链被触发后所指向的 URL。例如：

<A HREF="http://www.ecnu.edu.cn/about/page-5.htm">next page</A>

在 HTML 中还可使用相对 URL 来代替绝对 URL。例如要指向的另一 HTML 文件在同一目录下，只需简单地写为：

<A HREF="page-5.htm">next page</A>

如要指向上两级目录下的文件，可以这样写：

<A HREF="../../topic.htm">Return to topic</A>

通常超链只指向一个文件的头部，若要指向一个文件内的某一特定位置，就要用到超链标记的另一个属性 NAME，其格式如下：

<A NAME="超链名"> 超链文本及图像 </A>

3.4.2 页面链接的提取

页面链接的提取，只有当下载的文档是 txt/html 格式的页面时在有必要分析。页面的类型可由该网页头信息分析得出，有些站点返回的应答信息格式不完整，此时必须通过分析页面 URL 中的文件扩展名来判别类型。遇到带连接的标志如 <A>，<AREA>，<FRAME>等，就从标记结构的属性中找出目标 URL，并从成对的该标记之间提取正文作为该连接的说明文字（扩展原数据）。这两个数据就代表了该链接。

一个页面连接提取的过程总结如下：

1 判断文件的类型是否是 txt/html 如果不是则跳过，如果是继续分析。

2 读取文件利用正则表达式匹配的方法，寻找标签<a href= >，<area href=

> ，<base href= >，<frame src= >，<img src= >，<body background= >，<applet code= >等，记录下该 URL。

3 将记录下来的 URL 按照预定的格式统一完整。以备进行下一步的 URL 分析。

3.5 网页更新

早期的搜索引擎通常采用统一的周期性信息更新策略，即每隔一段（固定的）时间对索引信息库网页进行一次全面的信息更新维护，更新周期一般以星期或月为计量单位。但由于索引信息库规模巨大，例如 AltaVista、Lycos 等搜索引擎中都包含有几千万个 Web 页。如果在每一次更新过程中，对于每一个 WWW 文档都进行有效性验证，其工作量将十分巨大。假设一次有效性验证所需的时间为 0.001 秒，那么，要完成几千万个 Web 页的验证工作至少需要几天时间。事实上，许多文档的内容是很少改变的，没有必要不断地验证其有效性。为此，又出现了增量式信息更新方法。增量式信息更新方法的基本思路是：因为互联网中包含的大量网页的更新周期是不一致的，有的变化无常，有的十分稳定。因此应该以网页的变化周期作为进行有效性验证的依据，在每一次网页的更新过程中，只对那些最可能发生变化的网页进行更新。因为没有一个搜索引擎系统能够搜集到所有的 Web 页面，所以查全率很难计算，现在的搜索引擎更关心搜索结果的“重要性”（ Popularity）以及搜索结果的“时新性”（ Freshness），这就需要 Crawler 能够搜索到既重要又时新的网页。

目前，已存在的方案大致有如下几类，不同的更新策略会得到不同的更新结果。

1、统一更新策略[31]。Crawler以同样的频率f更新集合中的所有网页，而不考虑这些网页的改变频率。

2、个体更新策略[31]。不同网页其改变频率不同，Crawler根据个体页面的改变频率来更新个体页面。就是Crawler对每个页面都给定一个更新频率，且网页的改变频率与更新频率的比率对任何个体网页来说都是相等的。

在所有网页重要性相同的情况下，频繁访问改变太快的网页不能明显提高搜索效率，而应该把资源集中在那些可以跟得上其改变速度的网页上，这与个体更新的思想恰恰相反，因此一般来说，统一更新策略优于个体更新策略。但是如果采用统一更新策略，将会产生那些改变频繁的网页长期得不到更新的问题，因此，有人在（1）、（2）两种方案的基础上提出一个折中的更新策略：

假设所有网页的重要性都一样，设有 n 张网页，其改变频率的估算值分别为 f_1 ，$|G_1| \approx |G_2| \approx ... \approx |GN| = \Phi \gg P_2 \to T_2$ 为平均改变频率，根据网页的改变频率将网页分成两类：改变频率大于等于平均改变频率的网页分为一类 $F_1 = \{ f_i \mid f_i \geq f_a \}$；小于平均改变频率的网页分为一类 $F_2 = \{ f_i \mid f_i < f_a \}$；对于这两类网页分别计算出其平均改变频率 f_{a1} 与 f_{a2} ，然后按照这两个平均改变频率分别访问这两类网页。通过验证得出如下结论：对于所有网页重要性一样的分类访问策略，在系统资源允许的情况下，对改变频率大于等于平均改变频率的网页集合 F_1 应该以 5 f_{a1} 的速度来访问这些网页；而对于改变频率小于平均改变频率的网页集合 F_2 可以直接以频率 f_{a2} 访问这些网页，这是因为此类集合改变速度比较缓慢，用 f_{a2} 访问已经可以满足要求，同时也能减少系统开销。本文所应用的算法是在文献[32]提出的分类更新策略的基础上提出的，算法结合考虑了网页的重要性和时新性。

文献[32]提出的分类更新算法是基于所有网页集合的重要性相同的假设，实际上网页的重要性对于不同的主题，不同的时间是不同的[33,34]。对于重要的网页，即使其平均改变频率不是很快，考虑到其改变频率突然变快或者改变频率分布不均匀的情况，也需要提高其更新频率保证网页的时新性。而对于不重要的网页，即使它们改变比较频繁，也不需要提高其更新频率，反而降低其更新频率以节约系统资源。因此本文提出了一个提高重要网页的更新频率的更新算法：

对于网页集合 $S = \{p_1, p_2, ..., p_n\}$。求出其平均更新频率 $F_a(S)$，然后按照网页的重要性加权其更新频率将网页集合S 平均分为五类：很重要 S_1、较重要 S_2、重要 S_3、一般 S_4、不重要 S_5。因为网页的重要性符合 *Zipf* 分布[35]，所以按照重要性的不同来访问这五类集合的概率 $P(j)$ 应该符合如下条件：

$P(j)=C/j,$其中$1/C=\sum_{j=1}^{n}(1/j)$ $(j=1,2,...,N,N=5)$，根据公式算出对很重要集合S_1以平均概率的 2.2 倍左右访问，对较重要集合S_2以平均概率的 1.1 倍左右进行访问，对重要集合S_3以平均概率的 0.7 倍左右进行访问，对一般集合S_4以平均概率的 0.5 倍左右进行访问，对不重要集合S_5以平均概率的 0.4 倍左右进行访问。

对于重要性比较高的网页，采用这种算法中较高的更新频率，这样，重要性比较高的网页的时新性比采用原分类算法的时新性要高或者相等，即从理论上既保证了重要性又保证了时新性。但是，这样做的代价是重要性比较高的网页比采用原分类算法要消耗更多或相等的系统资源。重要性比较低的网页，采用两种分类算法中较低的更新频率，其优点是重要性比较低的网页会比采用原分类算法消耗更少的系统资源。总的来说，本算法的优点是比较重要的网页具有较高的时新性，但要消耗较多的系统资源；比较不重要的网页时新性较低，但是节约了系统资源。

3.6 本章小结

本章介绍了分布式网络爬虫的主要问题，包括：种子集合的选取，分布式策略的制定，多线程下载，网页分析和网页更新。

第4章 系统实现及实验评测

4.1 系统实现

本文描述的系统是基于 PC 机群的分布式网络爬虫，实现的环境是：7 台 PC 机，Fedora4 操作系统，机器的配置：CPU 3GHz，内存 2GB，硬盘 400G。下面的实现参考了文献[36~39]。

如图 4-1 所示是该系统总体设计图，由多台 PC 机组成，其中一台是控制节点，其它为爬行节点，控制节点负责维护所有结点的信息，对所有结点的信息进行同步，爬行节点上运行多个线程完成下载任务。

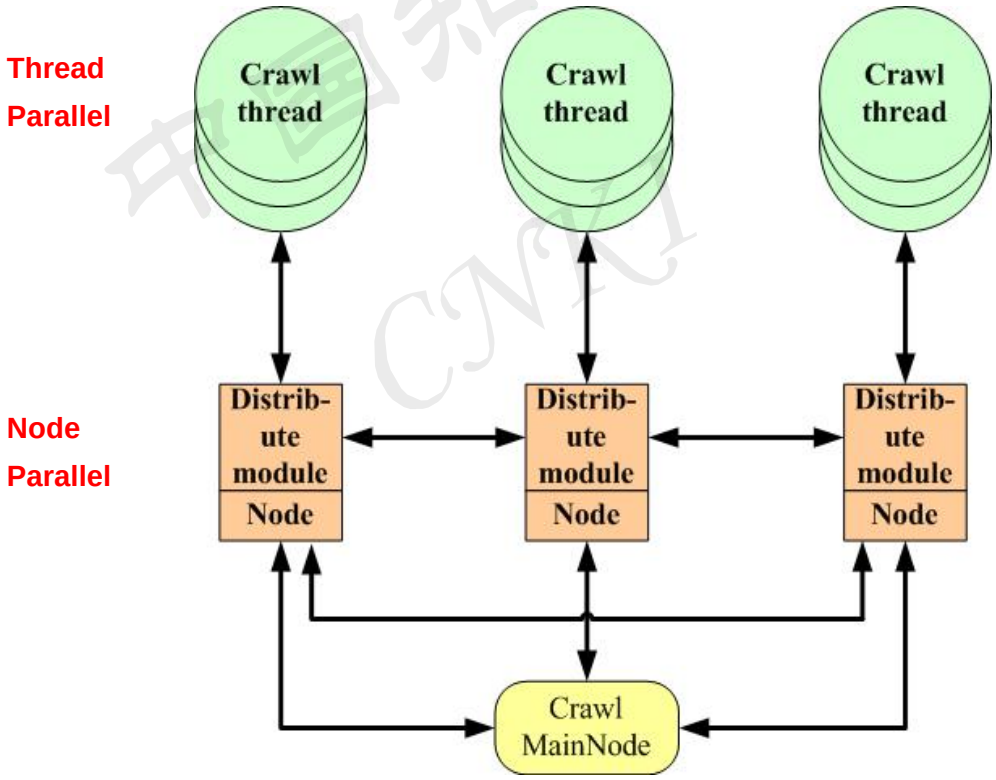

图 4-1 系统总体设计图

Figure 4-1 System architecture design

4.1.1 分布式任务分配的实现

首先在程序的配置文件中定义了一个选项，即该爬虫用于网站爬行还是普通爬行，在 main 函数的初始化阶段，解析配置文件。由于需要对每一个新解析出来的 URL 根据公式 3-1 计算 hash 值 i（0<i<逻辑结点数最大值），在结点由每个结点上面维护的一个映射表中得到具体的结点号 node_num。如图 4-2 所示，得到一个 URL 后交给 check_node()函数，在 check_node()中调用 hash 函数，由 hash()函数返回一个整数 i，这个 i 代表的是逻辑结点数，然后在 check_node()函数中查询二级映射的第二级列表，给出实际的结点号，最后判断，如果该结点号等于 local_node_num 则将该 URL 加入到 wait_list 中，否则加入到相应的 send_wait_list，每一个 send_wait_list 是和除了自己以外的每一个结点相对应的。

Send_check()函数实现的功能是当所有的 send_wait_list 中，长度超过一个阈值 SEND_LIST_MAX（宏定义）的时候，就触发 send()函数，该函数在一个线程中实现。

之所以选择设置 SEND_LIST_MAX，是因为如果分析得到一个 URL 就传送的话会在 TCP/IP 链路建立的过程中耗费大量的网络带宽，所以攒够了一定数量再发送。

每一个结点在一开始就运行了两个线程，如图 4-3 所示分别是：

1. Listen()线程 负责监听 SERVER_PORT 通信端口，接收其它结点传来的 URL 和控制节点的同步信息。每个结点在收到数据包后，调用解包函数，并按照优先级的高低向不同的等待队列中添加 url 类对象，解包函数的任务是收到 URL 字符串和它在网页的深度后调用 URL 类的构造函数重新生成 url 类对象。
2. Send()线程 负责监视 send_wait_list，每隔一定时间扫描一遍各个对列的长度，如果长度超过 SEND_LIST_MAX 则打包传送。打包函数为了减少结点间的通信，不传输整个 url 类对象而是仅仅传输 url 字符串和它在网页的深度。因此在传送前将 URL 类对象的 URLstring 和 depth 从 URL 类对象中提取出来，然后发送。

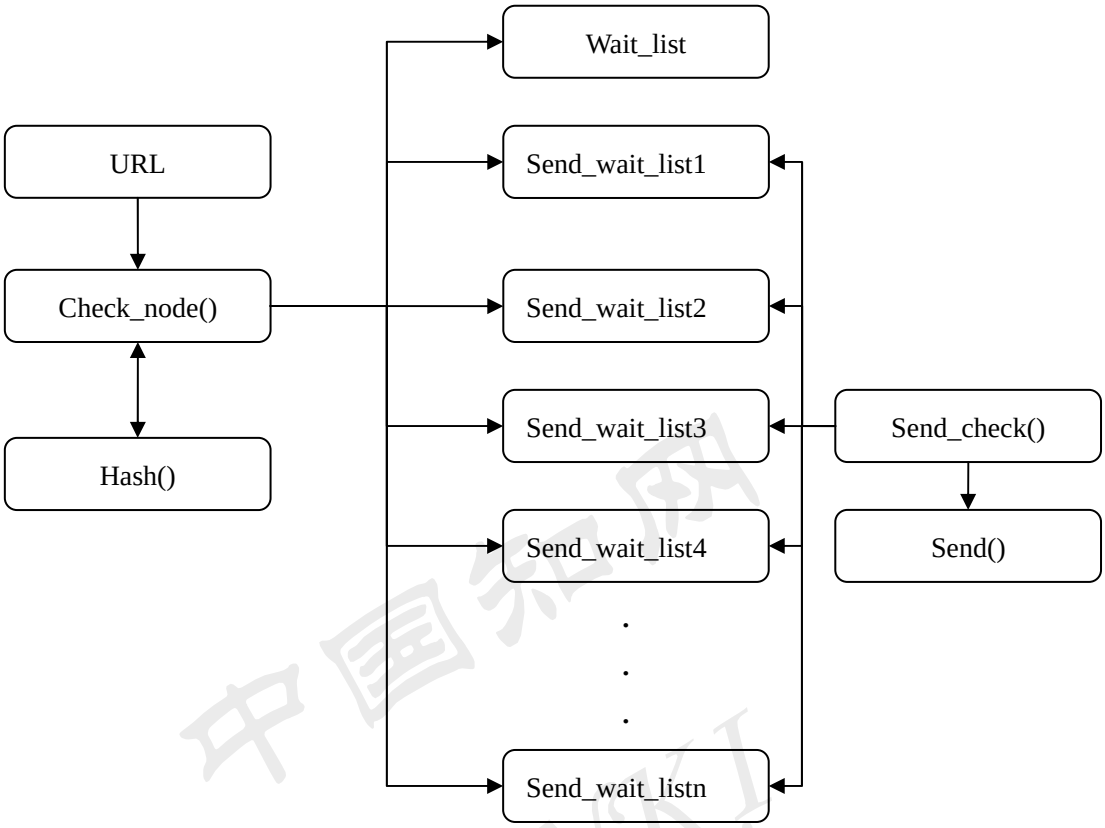

图 4-2 任务分配函数流程图

Figure 4-2 Task distribute fuction map

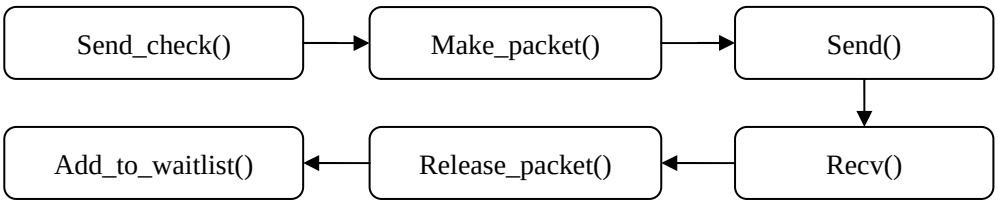

图 4-3 结点通信示意图

Figure 4-3 Node communication sketch map

4.1.2 单结点下载任务的实现

图 4-4 显示的是单个结点在得到 URL 数据后的处理过程，数据的来源有三个地方，自身下载线程，控制节点控制信息，其它结点。得到 URL 后要判断，该 URL 是否归本结点处理，该 URL 是否已经下载，如果该 URL 归本结点处理，且没有下载过，就加入到 unvisited_table 中，然后再选择一个闲置线程从 unvisited_table 读取一个 URL 进行下载，同时调用 DNS 分析模块，在取得网页内容后，进行网页解析，提取 URL，然后对提取出来的 URL 进行分析，如果遇到过则丢弃，否则计算是否归本结点处理，如果不归本结点处理，则发送到相应的结点。

接着来看单个结点的处理流程，具体程序中是如何实现的，如图 4-5 所示：在程序启动后首先调用的是初始化函数，初始化函数完成的是，数据结构空间的申请，参数的初始化，并调用配置文件解析函数。并把起始 URL 加入到 WaitList 中。

然后进入一个循环，在循环的开始，最先调用的是 listRead()函数，这个函数实现的功能是操纵等待队列，由于前面所介绍的原因，等待队列是不可能完全存在在内存中的，所以这个函数其实需要调用很多其它的等待队列处理函数，比如把等待队列切分，把近期不会用到的、或是优先级不高的部分放在内存中。

接下来的是fetch()函数，这个函数完成的任务是下载上一步得到的URL，这里的下载是并行进行的，所以还涉及到进程的同步通信，还有DNS解析也是在这里完成的，该系统用的是ADNS这个DNS解析库[40]，来实现的驻留内存的DNS Cache。

然后是 HTML 的解析部分，FetchRead()从 socket 中读数据，ParseHtml()解析网页，这个函数首先解析的是 HTML 的头信息，然后才是正文，最后调用 ExtractUrl()，提取 URL（参照 3.4 网页分析）。

然后要做的就是分析 URL 是否是下载过的，还是已经存在在等待队列中，这一部分的实现参照 2.2.1.3 节的讨论，由于这项操作会很频繁的进行，所以为了实现程序高速运行，该系统把下载过的 URL 对其 host 进行散列存储，这样比较算法的时间复杂度会下降很多。

分析完 URL 之后就是上一小节所介绍的调用 check_node()函数，最后返回到循环的开始。

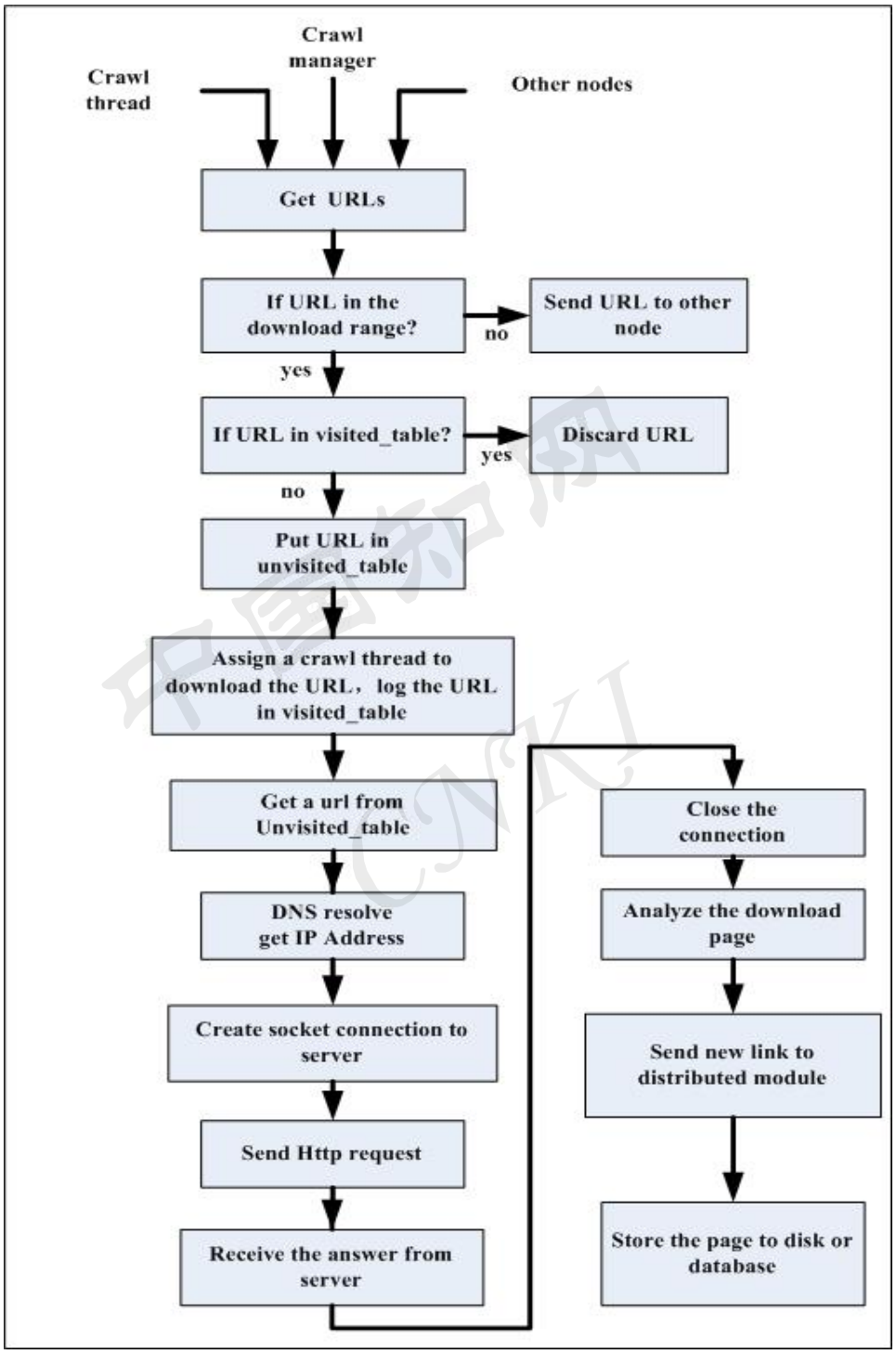

图 4-4 URL 数据流程图

Figure 4-4 URL data flow chat

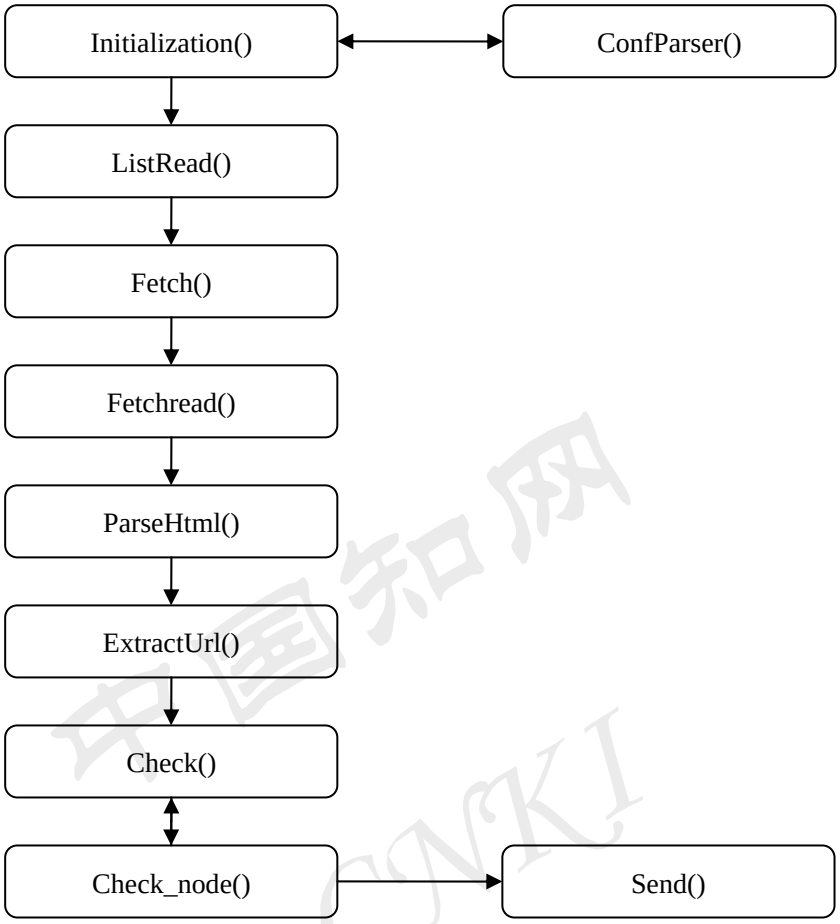

图 4-5 程序流程图

Figure 4-5 Overview flow chart

4.2 系统评测

爬虫的评测从两个方面进行，首先要测的是这个爬虫的负载平衡，通信负载，单个结点的下载速度和分布式爬虫中的单个结点的速度的比较。然后作为本爬虫的一个应用对一个网站进行全站爬行，并测试 3.2.5 节中讨论的算法。

4.2.1 普通爬行评测

1. 单机爬行测试：

参数设置：

表 4-1 参数表

Table 4-1 Parameter table

参数	参数值
下载线程数	200
Dns 线程数	5
网站爬行深度	15
同一服务器访问间隔	35s
初始 URL	www.163.com

爬行结果：爬行 10 小时，下载网页 1 176 804 个，大小 26.16GB

2. 2 个结点并行爬行测试：

参数设置：

表 4-2 参数表

Table 4-2 Parameter table

参数	参数值
下载线程数	200
Dns 线程数	5
网站爬行深度	15
同一服务器访问间隔	35s
初始 URL	www.163.com
SEND_LIST_MAX	500
Send_check 查询时间	0.1s

爬行结果：爬行 10 小时

结点 1：下载网页 901 355 个，大小 21.34GB

结点 2：下载网页 681 645 个，大小 15.94GB

结点 1 向结点 2 传送了 4 645 344 个 URL，结点 2 向结点 1 传送了 3 895 462 个 URL。

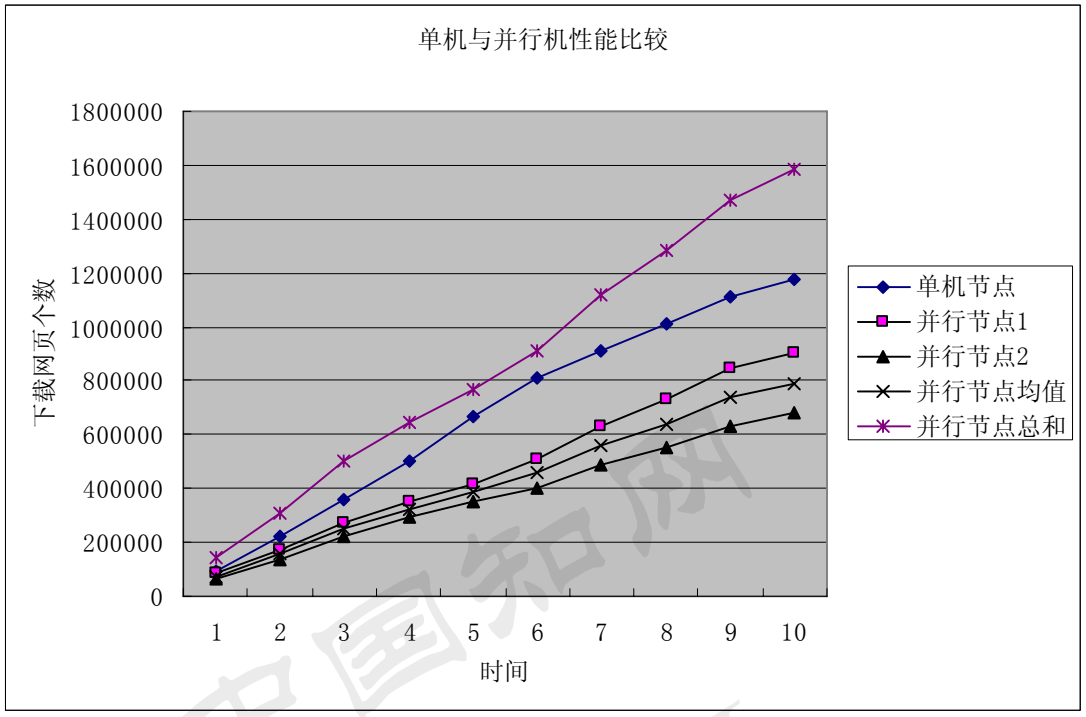

图 4-4 单机与并行结点性能对比

Figure 4-4 The performance of stand-alone and distribute

本测试是从晚上 10：00 开始到次日上午，单结点得到的结果比较理想。并行测试中，单个结点的效率同单机版相比下降了一些，但是这个下降是必然而且在正常的范围之内，因为两个结点同时爬行的时候是公用同一条对外网络接口，网络带宽成了并行爬行的主要瓶颈，两个结点的运行效率还是比单个结点高出了将近 40 个百分点，相信网络环境的改善能够大大的提升并行网络爬虫的工作效率。

4.2.2 网站全站爬行评测

本评测的目的是检测该爬虫在网站全站爬行的应用效果，本试验选择的网站是 www.163.com。

1. 单机爬行测试

参数设置：

表 4-3 参数表

Table 4-3 Parameter Table

参数	参数值
下载线程数	200
Dns 线程数	5
网站爬行深度	15
同一服务器访问间隔	35s
初始 URL	www.163.com

爬行结果：

爬行时间 10 小时，下载网页 32 068 个文件 大小 725M

2. 2 个结点爬行测试

参数设置：

表 4-4 参数表

Table 4-4 Parameter Table

参数	参数值
下载线程数	200
Dns 线程数	5
网站爬行深度	15
同一服务器访问间隔	35s
初始 URL	www.163.com
SEND_LIST_MAX	500
Send_check 查询时间	0.1s

爬行结果：

爬行时间 10 小时

结点 1：下载网页 28 032 个，大小 652MB

结点 2：下载网页 26 594 个，大小 602MB

结点 1 向结点 2 传送了 4 645 344 个 URL，结点 2 向结点 1 传送了 3 895 462 个 URL。

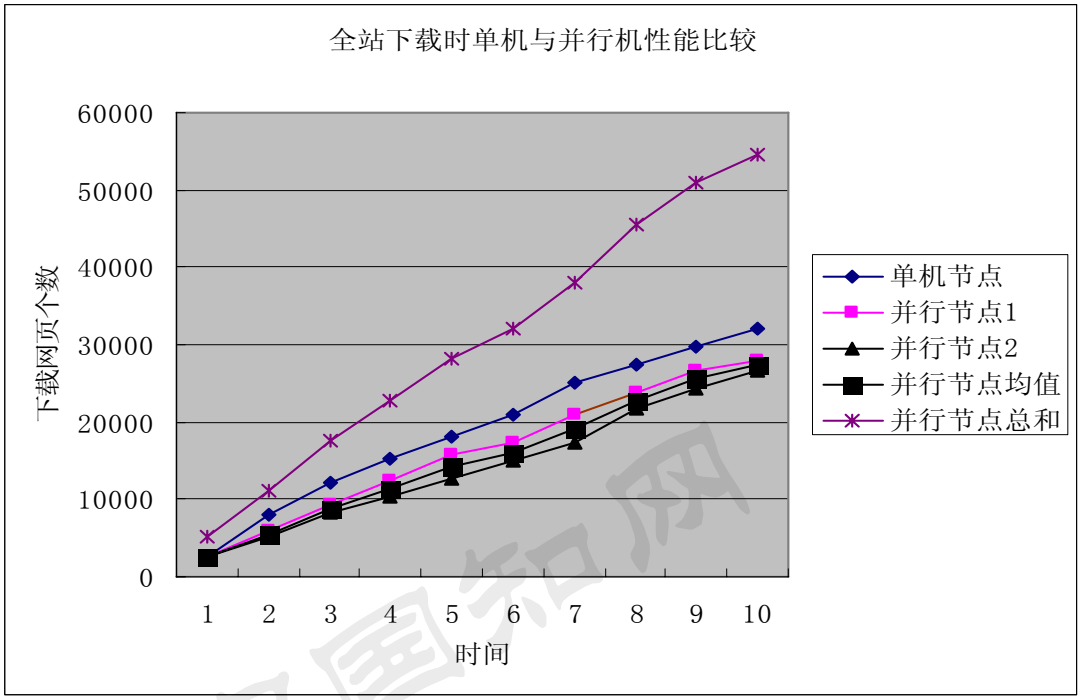

图 4-4 网站全站下载单机与并行结点性能对比图

Figure 4-4 The performance of stand-alone and distribute at site download

本测试在白天进行，可以得出一些结论，网站爬行由于频繁的访问一个网站，多线程的并行度会下降很多，因此爬行的速度很慢，但是由于本文在 3.2.5 节的算法的实施，使得结点间的并行度下降的不是很大。

4.2.3 本章小结

本章对本文实现的分布式网络爬虫进行了测试，包括普通测试和网站全站下载测试，而这些测试又包括了单机测试和并行测试。通过测试，看到了本系统的优势和不足，这样对未来的工作有一定的指导。

结论

本文研究了如何建立一个分布式的网络爬虫，探讨了分布式网络爬虫的关键技术。本文采用的构架是 N 台普通 PC 机，其中一台是控制节点，其余为爬行节点，由控制节点负责监控整个爬虫的工作，各个爬行节点协同工作下载网页。

分布式带来了一些问题，比如，各个结点如何保证下载的网页没有重复，如何保证尽可能多的网页都被下载，而不会因为某个结点的丢弃而丢弃，结点如何添加和删除而不影响爬虫的工作，这些归结起来都是分布式网络爬虫任务分配的问题，为了解决这个问题，本文讨论了目前分布式系统的任务分配都使用了什么方法，任务分配的粒度大小的选择，和分配函数的选择，最后提出了二级映射的方法来解决这个问题，首先本文选择网页的主机归属作为任务分配的粒度，把所有的网页按照主机归属散列到 N 个逻辑结点中去，然后再由逻辑结点映射到物理结点上，这种方法使得分布式的任务分配有了一个很好的解决，保证了结点间负载平衡，结点间的通信负载和结点与主机的通信负载也处在一个较为理想的程度，并且能够防止一个结点的死机影响整个分布式网络爬虫的运行，拥有了一定的健壮性，然后又能方便的添加删除结点，使得可扩展性也有了保障。

本文在最后设计了一个分布式网络爬虫的系统。该系统可以灵活的配置参数，并具有健壮性，可扩展性，然后应用这个系统实现了网站全站下载的功能，试验显示，并行网络爬虫的下载速度较单机爬虫有了很大的提高，可以作为一个可制定的中型规模的网络爬虫投入使用。

网络爬虫是搜索引擎的信息来源，没有好的网络爬虫，搜索引擎将会是无源之水。因而，网络爬虫是搜索引擎中非常重要的一个子系统。本文主要研究了网络爬虫的关键技术。并实现了一个分布式的网络爬虫，虽然性能上较单机爬虫有了很大的提高，但是，本系统在智能性和规模上还有很多需要改进的地方。

参考文献

1 郭祥昊，李蕾. 中文智能搜索引擎. http://www.ccidnet.com

2 D. Eichmann. The RBSE Spider-Balancing Effective Search Against Web Load. In Proceedings of the First International World Wide Web Conference, 1994:113~120

3 O. A McBryan. GENV and WWW: Tools for Taming the Web. In Proceedings of the First International World Wide Web Conference 1994:79~90

4 B. Pinkerton. Finding What People Want: Experiences with the Web Crawler. In Proceedings of the First International World Wide Web Conference 1994:54~65

5 潘春华, 常敏等. 面向Web的信息收集工具的设计与开发. 计算机应用研究, 2002,(6):144~148

6 Barroso, L. A. Dean, J. Holzle. Web search for a planet: The Google cluster architecture. Digital Object Identifier, 2003:107~117

7 M. Burner. Crawling towards Eternity: Building an Archive of the World Wide Web. Web Techniques Magazine, May 1997, 2(5)

8 A. Heydon, M. Najork. Mercator: A Scalable, Extensible Web Crawler. In Proceeding of the 10th International World Wide Web Conference, April 2001

9 http://www.robotstxt.org

10 Andrei Z. Marc Najork. Efficient URL caching for World Wide Web crawling. ACM press. 2003 679~689

11 RFCs.2004. Domain names. http://www.faqs.org/rfcs/rfc1034.html

12 RFCs.2004.Hypertext Transfer Protoco HTTP/1.0. http://www.faqs.org/rfcs/rfc1945.html

13 RFCs.2004. Hypertext Transfer Protocol HTTP/1.1. http://www.faqs.org/rfcs/rfc2616.html

14 C E.Eugene Schultz 著. 段海新等译. 网络安全事件响应. 北京,人民邮电出版社，2002:25~50

15 冉晓旻. 信息安全策略. 信息网络安全. 2003,04:50~70.

16 Design and Implementation of a High-Performance Distributed Web Crawler Vladislav Shkapenyuk Torsten Suel Technical Report TR-CIS-2001,03: 15~19

17 Sergey Brin, Lawrence Page. The Anatomy of a Large-scale Hyper textual Web Search Engine. In Proceedings of the First International World Wide Web Conference, 1998: 107~117

18 http://www.google.com

19 http://www.baidu.com

20 http://www.yahoo.com

21 Grub, a distributed crawling project. http://www.grub.org, 2004. GPL software

22 A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. World Wide Web Conference, 1999 (4): 219~229

23 V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, IEEE CS Press. 2002: 357~368

24 T. Suel and Jun Yuan. Compressing the graph structure of the Web. In Proceedings of the Data Compression Conference DCC, IEEE CS Press, 2001. 213~222

25 Demetrios Zeinalipour-Yazti, Marios D. Dikaiakos. Design and implementation of a distributed crawler and filtering processor. In Proceedings of the fifth Next Generation Information Technologies and Systems (NGITS), volume 2382 of Lecture Notes in Computer Science, Caesarea, Israel, 2002: 58~74

26 Junghoo Cho, Hector Garcia-Molina. Parallel crawlers. In Proceedings of the eleventh international conference on World Wide Web, Honolulu, Hawaii, USA, ACM Press, 2002: 124~135

27 K. Kemidy, Chales F. Bender, John W. D. Connolly, John L. Hennessy, Mary K.Vernon, Larry Smarr, "A Nationwide Parallel Computing Environment", Communications of the ACM, 1997. 10

28 L. Lamport, "How to Make a Multiprocessor Computer That Corredtly Executes Multiprocess Programs", IEEE Transactions On Computers, 1979: 690~691

29 D. Culler, Jaswinder Pal Singh, an Anoop Gupta. Parallel Computer Architecture A Hard ware/Software Approach. 1997

30 http://www.shanxiwindow.net/teaching/htmlbook/

31 Junghoo Cho, Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler [D]. Stanford University, 2002

32 文坤梅,卢正鼎等. 搜索引擎中页面分析策略的分析与改进. 华中科技大学学报(自然科学版) , 30 (12):3~5

33 王继民. 国内综合性搜索引擎时新性的计算. 计算机工程与应用, 2003, (21) :47~49

34 Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers.ACM Transactions on Database Systems, 28(4), December 2003

35 Ricardo Baeza-Yates, Felipe Saint-Jean, Carlos Castillo. Web structure, dynamics and page quality. In Proceedings of String Processing and Information Retrieval (SPIRE), Lisbon, Portugal, Springer LNCS, 2002:117~132

36 R. Miller and K. Bharat. Sphinx: A framework for creating personal, site-specific web crawlers. In Proceedings of the seventh conference on World Wide Web, Brisbane, Australia, April 1998

37 R. Miller. Websphinx. A personal, customizable web crawler. http://www-2.cs.cmu.edu/rcm/websphinx, 2004. Apache-style licensed, open source software

38 D. Eichmann. The RBSE spider: balancing effective search against web load. In Proceedings of the first World Wide Web Conference, Geneva, Switzerland, May 1994

39 Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. ACM Transactions on Database Systems, 28(4), December 2003

40 Ian Jackson. ADNS. http://www.chiark.greenend.org.uk/~ian/adns/, 2002

哈尔滨工业大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《分布式网络爬虫技术的研究与实现》，是本人在导师指导下，在哈尔滨工业大学攻读硕士学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签字：　　日期：　　年　月　日

哈尔滨工业大学硕士学位论文使用授权书

《分布式网络爬虫技术的研究与实现》系本人在哈尔滨工业大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归哈尔滨工业大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解哈尔滨工业大学关于保存、使用学位论文的规定，同意学校保留并向有关部门送交论文的复印件和电子版本，允许论文被查阅和借阅，同意学校将论文加入《中国优秀博硕士学位论文全文数据库》和编入《中国知识资源总库》。本人授权哈尔滨工业大学，可以采用影印、缩印或其他复制手段保存论文，可以公布论文的全部或部分内容。

作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日

哈尔滨工业大学硕士学位涉密论文管理

根据《哈尔滨工业大学关于国家秘密载体保密管理的规定》，毕业论文答辩必须由导师进行保密初审，外寄论文由科研处复审。涉密毕业论文，由学生按学校规定的统一程序在导师指导下填报密级和保密期限。

本学位论文属于　　保密□，　在　　年解密后适用本授权书

不保密□

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日

致谢

值此论文完成之际，谨向给予我无私帮助的老师和同学们致以最诚挚的谢意！

感谢我的导师王晓龙教授，他不仅在学术上对我悉心指导，而且在生活上也关怀备至，我的每一分进步都离不开他的心血。王老师严谨的治学态度，勤奋的工作精神永远是我学习的榜样。很高兴在王老师的指导下度过这两年宝贵的时光。

感谢徐志明老师，在一起工作的日子里，徐老师在学术上的指导，生活中的关怀，使我置身在一个非常愉快的工作环境中。和徐老师每一次交流都使我获益匪浅，感谢徐老师的教导和帮助。

感谢博士研究生朱鲲鹏师兄、赵玉茗师姐、孙广路师兄、董启文师兄、赵健师兄、王强师兄、刘桃师姐。各位师兄师姐在课题中对我提出了各种宝贵的建议，使我能够顺利完成课题。

感谢哈尔滨工业大学自然语言处理研究室的所有老师和同学们，与你们在一起的时光我终身难忘。

最后，感谢我的亲人们，尤其要感谢我的父母，正是他们的关爱与支持才使得我有了不断前进的动力。