高性能网络爬虫:研究综述

周德懋 李舟军
(北京航空航天大学计算机学院 北京 100191)

摘 要 网络爬虫是一种自动下载网络资源的程序,是搜索引擎的基础构件之一。系统地介绍了网络爬虫的工作原理和发展现状,详细地阐述了一个高性能、可伸缩、分布式的网络爬虫的系统架构和所面临的关键问题。

关键词 网络爬虫,高性能,可伸缩,分布式

Survey of High-performance Web Crawler

ZHOU De-mao LI Zhou-jun
(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract Web Crawlers, one of basic components of Search Engine, are programs to download resources from Internet. We illuminated the work theory of the Web Crawlers, and its development, and how to design a high-performance, scalable, distributed Web crawler, including the faced key problem.

Keywords Crawler, High-performance, Scalability

1 引言

信息社会的飞速发展使互联网的容量达到一个空前的高度。Google宣称它们索引的网页数目已达到10000亿[10],中国的网页规模也超过了100亿[11],这对搜索引擎提出了更高的要求。搜索引擎的性能指标主要有3个:首先考虑的是规模的大小,只有规模达到一定的数量级,搜索结果才能更好地满足用户;其次是性能,搜索引擎必须在一个较短的时间内完成对目标网络的信息搜集,同时能够在用户可容忍的时间段内完成搜索结果的反馈;最后是搜索的质量,能够去掉信息重复的网页,对一些无用信息进行过滤,能够准确返回用户想要的结果。

作为搜索引擎的基础构件之一,网络爬虫(Crawler)直接面向互联网,它是搜索引擎的数据来源,决定着整个系统的内容是否丰富、信息能否得到及时更新。它的性能表现直接影响整个搜索引擎的效果。Crawler的工作原理如下:从一个初始URLs集(称为种子URLs)出发,从中获取一个URL,下载网页,从网页中抽取所有的URLs,并将新的URLs添加到URLs队列中。然后,Crawler从队列中获取另一个URL。重复刚才的过程,直到Crawler达到某种停止标准为止。

Crawler的工作原理如此简单,然而设计一个高性能的网络爬虫却是一件相当有挑战性的工作,不管是在学术界还是工业界,对它的研究和改良从未间断过。一个高性能的Crawler需要从以下几个方面来考虑:(1)可伸缩性。能胜任海量数据的抓取,并可通过增加硬件资源使性能得到线性提高。(2)分布式。集中式的Crawler架构已经不能满足目前互联网的规模,因此支持分布式的爬行,处理和协调好各结点之间的交互,也是一个重要议题。(3)"礼貌"爬行。Crawler不能在短时间内大数据量地集中访问同一个主机下的网页,否则会影响普通用户对其的访问,进而可能被对方限制访问。(4)可定制性。可根据不同的爬行任务(例如Blog,BBS)和特定的主题定制相应的功能模块,使功能插件化,打造个性化Crawler。

2 研究现状

斯坦福大学设计了用于Google的爬虫[6]。早期的Google爬虫系统由5个模块处理不同的任务。一个URL服务器从磁盘文件读URL列表并将其转发到Crawler上。每个Crawler单独运行在一台机器上,采用单线程异步IO方式,一次维持300个连接并行爬行。Crawler将网页传输到存储服务器上压缩并保存。索引进程从HTML页面中抽取链接并存放在不同的文件中。一个URL解析器读取这些链接文件并转化为绝对路径,由URL服务器读取。后期Google的改进主要有:(1)采用自有的文件系统(GFS[3])和数据库系统(BigTable[5])来存取数据;(2)采用MapReduce[2]技术来分布式处理各种数据的运算。

康柏系统研究中心的Allan Heydon和Marc Najork设计了名叫Mercator[17]的爬行器。系统采用Java的多线程同步方式实现并行处理,并加入了很多优化策略如DNS缓冲、延迟存储等以提升爬行器运行效率。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

到稿日期:2008-09-28 返修日期:2009-02-11 本文研究得到国家自然科学基金项目(60573057,90718017)资助。

周德懋(1984-),男,硕士研究生,研究方向为分布式计算、文本挖掘,E-mail:zdmeng@163.com;李舟军(1963-),男,教授,博士生导师,研究方向为高可信软件技术、安全协议的形式化验证、数据挖掘与文本挖掘。

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 是否已经访问过。

Internet Archive[7] 的每台 Crawler 同时对 64 个站点进行爬行，每个站点被唯一分派到一个 Crawler 上。Crawler 从磁盘上读取 URL 列表，采取异步 IO 方式下载网页，并抽取链接。如果该链接属于本机抓取，则放入待抓取列表，存到磁盘上，并周期性地传送到其它 Crawler 上。

UbiCrawler[8] 项目是一个高性能的爬虫，主要侧重于完全分布性和高容错率。它的主要特性包括：平台独立性、良好的伸缩性、高效的分配函数、各功能模块的完全分布式、没有单点故障的问题。

IRLBOT[9] 是 TAMU 开发的大规模网络 Crawler，它们宣称已经抓取了 60 亿网页。该爬虫能胜任 100 亿级网页爬行，可伸缩性很强，在“礼貌”爬行和反垃圾页面上做了很多工作。

北大天网[13] 是国内高性能网络爬虫的先行者，它的架构经历了集中式向分布式的改进，能够胜任 10 亿级的网页搜索，其基于站点的两阶段哈希机制有效地解决了搜索过程中 Crawler 动态加入和退出的问题。

3 网络爬虫的系统架构

网络爬虫的结构主要分为以下几个部分：(1)下载模块，(2)网页分析模块，(3)URL 去重模块，(4)URL 分配模块，如图 1 所示。

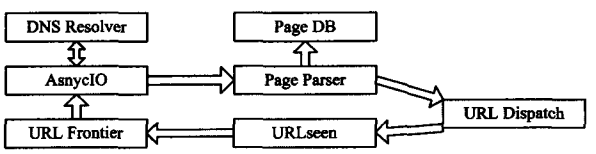

图 1 系统架构图

3.1 下载模块

下载模块负责抓取网页，是整个系统的基本与关键部分，直接影响爬行效果。该模块一般包含 3 个子模块：(1)下载线程；(2)Host 控制子模块；(3)DNS 解析子模块。

3.1.1 下载线程

下载线程主要通过 HTTP 协议与 Web 服务器进行通信。采用 socket 方式下载的网络编程模型主要有同步 IO、非阻塞 IO、异步 IO。同步 IO 采用每一线程对应每一连接，编程简单，且性能随着 CPU 个数的增加而呈线性增加，但单个 CPU 的扩展性差，随着连接的增多线程的切换将是一个很大的开销。非阻塞 IO 性能较之同步 IO 有一定提高，适合中等规模的网络应用。异步 IO 是操作系统专门为之优化的一种模式，具有扩展性强、性能优越的特点，是高性能网络编程的最佳选择，但编程复杂，难度较大。

早期的 Google 采用单线程异步 IO 方式，一次维持 300 个连接并行爬行。Mercator 采用多线程同步下载。Internet Archive 采用单线程异步 IO，有 64 个异步下载端。UbiCrawler 仅采用 4 个线程进行同步抓取。而 Polytechnic[14] 维持了 1000 个异步连接。天网采用 150 个同步线程。

需要指出的是，即使是异步方式，也不能维持大量的连接，否则会浪费宝贵的未分页内存和 CPU 内核切换时间。从上面可以看出，依照具体的环境，64 到 300 个连接比较适中。

3.1.2 Host 控制子模块

该子模块的作用是防止抓取端在短时间内大量访问同一主机下的页面，造成类似于拒绝服务攻击的效果而封掉 IP。其基本策略是某一时刻保证只有一个抓取线程访问某一特定主机，并且在一定时间内不会再次访问。其它还要注意的问题有：控制同一个网站的数据流量，并实时统计；遵循 Robots 协议，不下载禁止访问的目录。

3.1.3 DNS 解析子模块

访问一个网页首先得知道其 IP 地址，在 Berkeley Sockets API 里是用 gethostbyname (const char* name)这个函数获取相应主机的 IP 地址。DNS 解析是网络爬虫的瓶颈之一，因为有些域名请求要经过很多层服务器才能解析到，或者因解析服务器的忙碌而超时。而 gethostbyname(const char* name)函数是同步的，即使是在多个线程里调用这个函数，它们都会阻塞在一起，依次等待结果的返回。有文献[17] 指出，DNS 查询占用整个爬行的时间高达 70%。解决的方法有两种：一是提供 DNS 缓存，二是建立异步 DNS 查询模块。在实际应用中一般会综合这两种方法。DNS 解析子模块可以单独拿出来放在一台 Server 上做成 DNS 服务器，这样可以采用更大的缓存和更多的查询线程。

3.2 网页分析模块

网页分析主要是内容分析和链接抽取。网页中有很多不同的编码格式，这些格式来自不同的文本(简体中文、繁体中文、英文等)。这些不同的文本信息会影响到后续的正文抽取和分词等模块。网页分析中需要考虑到这类问题。HTML，XML 网页除了标题和正文以外，会有许多版权信息、广告链接以及公共的频道链接，这些链接和文本一般没有太大的价值，在提取网页内容的时候，需要过滤这些无用的链接。对于 DOC，PPT，XLS，PDF 等带格式的文件，网络爬虫都要提取出文件里的纯文本内容。对于多媒体、图片等文件，一般是通过链接的锚文本(即链接文本)和相关的文件注释来判断这些文件的内容。另外，许多多媒体文件中有文件属性，考虑这些属性也可以更好地了解文件的内容。

网页分析模块主要包含预处理子模块和信息抽取子模块。

3.2.1 预处理子模块

此模块主要负责网页分析之前对网页内容、编码、类别等做相应分析前的处理工作：(1) 编码转换，其作用是对网页内容进行编码转换工作，将其他种类的编码类型转换成 GBK 形式的类型，同时将繁体字转换成简体字；(2) CSS 处理模块，用处是从网上抽取网页中相关的 CSS，JS 以及 Title，Meta 等信息；(3) DOM 解析模块，其作用主要是根据网页 HTML 标签以及 JS 等信息构造 DOM 分析树，并为后续网页分析提供分析依据。

3.2.2 信息抽取子模块

当预处理完成后，程序将进入网页分析和信息抽取过程，这包含标题、正文等信息的抽取，生成摘要，以及对网页的分类：(1) 抽取网页中的标题、正文等有用信息；(2) 生成摘要，主要用来对网页中的正文信息进行标识；(3) 根据网页中相关文本以及结构等特征信息，对网页进行分类处理，识别出内容页、列表页、错误页、登录页等等。这些信息将会传给 URL

数据库，并进行相关的数据统计。

3.3 URL去重模块

在下载的过程中，不可避免地会遇到重复的链接，如何消除这些重复的链接，是个很复杂的议题。URL的去重可以说是爬虫系统中最重要的一部分，直接影响爬行效率和效果。目前主流网络爬虫的URL去重机制主要有两种方式：(1)完全内存方式；(2)基于磁盘的缓存方式。

完全内存方式就是指计算URL的Hash值，一般用4至6个字节表示，这样10亿个网页的URL就需要5到8个G的内存。另一种方式是开一个大数组进行按位验证，这样只需前者八分之一的内存，但有可能误判，且误判率随着爬行规模的增大而提高。

基于磁盘的缓存方式则是将大部分数据放在磁盘上，内存里存放一个缓存，然后根据策略更新缓存。由于磁盘的速度比内存慢一个数量级，一旦所查URL不命中缓存，就必须在磁盘中进行查找，从而大大影响效率。

Internet Archive，Google，UbiCrawler采用基于内存的方式，主要有平衡树方法和Bloom filter。Bloom Filter是一种空间效率很高的随机数据结构，在搜索引擎领域用处极大。它利用位数组很简洁地表示一个集合，并能判断一个元素是否属于这个集合。Bloom Filter的这种高效是付出一定代价的：在判断一个元素是否属于某个集合时，有可能会把不属于这个集合的元素误认为属于这个集合(false positive)。因此，Bloom Filter不适合那些“零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter通过极少的错误换取了存储空间的极大节省。IRLBOT，Mercator，Polytechnic采用基于磁盘缓存的方式，都用到一种滞后更新策略，基本原理是在内存里缓存一部分数据，待数据增长到一定规模后，再和磁盘的数据合并更新，大大减少了随机读取磁盘的次数。

3.4 URL分配模块

抓取的效率主要依赖于硬件资源、网络的带宽以及程序执行效率等。普通单处理机系统受限于CPU的处理能力、磁盘存储的容量，不可能具备处理海量信息的能力，这就要求Crawler支持分布式协同工作。商业化的大型搜索引擎的网络爬虫都部署在多个IDC(数据中心)机房，进行分布式爬行。每个机房又有多个实例在协同爬行。一般来说，每个IDC的爬虫会负责自己的IP段范围内的网页。同一IDC内的不同爬虫实例会根据一定的策略爬行各自的网页。一般说来，Crawler的分布式爬行[12]可分为内部分布式爬行和外部分布式爬行。

(1) 内部分布式爬行。所有的爬行进程在同一个本地网络上运行并通过一个高速连接(如LAN)进行通信，从远程Web站点下载网页时都利用相同的本地网络。采用这种方式，硬件资源扩展方便，几台PC就能增大磁盘容量，提高I/O吞吐量，做成一个小机群，性价比较高。此时，瓶颈主要是在网络出口带宽上。

(2) 外部分布式爬行。当并行爬行的不同爬行进程在通过Internet相连的地理位置较远的不同地区运行时，则称这种爬行为外部分布式爬行。它的优势是网络带宽较富裕，可以就近爬行周围的Web站点，速度较快。在这种情况下，重要的是确定不同地理位置的爬行进程间进行通信的频率和数量。因为进程间的带宽存在限制，有时甚至是拥塞不堪而导致堵塞。

分布式爬行的主要问题是当多个爬行节点并行下载网页时，不同的节点可能会多次下载同一个网页。为了避免这种交叉，并同时提高网页下载质量，并行节点之间应该进行充分的通信，在网页下载上达成协调，以便并行、一致、高效率地下载网页。

URL分配模块主要考虑两个问题：(1)在节点间划分URL的策略，即如何分配下载任务；(2)优化性能，比如负载均衡、协同工作的开销等。

目前一般有两种分配模式可以参考：

(1)静态分配模式。各节点按事先规定的URL范围独立下载。若遇到不属于本节点的URL，有3种处理方法：1)丢弃；2)下载；3)传送到URL隶属的节点。静态分配模式的优点是配置比较简单。关键在于如何划分URL范围，有效利用各个节点的资源。

(2)动态分配模式。由一个统一的URL管理器统一调度，根据各节点的情况动态地分配URL。该模式的优点是能做到负载均衡，使各个节点下载最大化。缺点也是很明显的，要有一个专门的URL管理器，增大了成本和配置难度。URL管理器需要与各节点保持高速实时通信，存在单点故障。由于要存放所有节点需要的URL集合，当下载规模增大时，本身也是一个瓶颈。

4 设计网络爬虫的关键问题

4.1 可伸缩性

面对网络上数以万亿计的网页，使用有限的资源运转一个高性能、可伸缩的Crawler是一个首要任务。完成这项任务通常有3点要求。首先，采用的算法和数据结构要能够支持Crawler处理海量的网页。其次，在有限资源下，爬行的平均速度必须维持一个较高的水平。然而对于大多数Crawler来说，随着下载页面的增多，对URL的唯一性检测、DNS Cache的查询将会缓慢很多，进而影响下载速度且要求更多的资源。第三，在添加硬件的情况下，性能能够得到线性增长。

网络爬虫要长时间运行，需要有很好的稳定性、良好的内存管理和资源调度，同时需要考虑广域网中各种异常情况。Crawler程序是大型搜索引擎中很脆弱的部分，因为它与很多外部的Web服务器打交道，而这些服务完全在系统的控制之外。一个页面出现问题就很可能导致Crawler程序中止、崩溃或其它不可预料的行为。因此，访问Internet的Crawler程序应该设计得非常强壮，要充分考虑各种可能遇到的情况，让Crawler在遇到各种情况时可以采取相应的处理措施。因此，错误处理、超时重传、丢弃等各种情况的处理都必须考虑到。在效率方面，由于网络爬虫可以看成是一个生产者-消费者模型，如何使各个模块协调工作，以达到最佳性能，也是一个严重的挑战。

4.2 提高下载质量

由于网络信息的爆炸，网络上的数据已经是PB级(1PB=100万GB)。在有限的时间内，网络爬虫只能搜集到有限的网页，因此我们总是希望尽量搜集到比较重要的网页，或者至少不要漏掉那些很重要的网页。

网络爬虫的目标是抓取互联网上所有有价值的网页。哪

些网页有价值，如何抓取这些网页，这些问题需要对网页的质量有一个较为全面的评价。而评价标准主要依赖对互联网和用户需求的理解。其中，主要包括以下几个方面的问题：

①网页之间的链接关系。当一篇网页被很多重要的网页所指向，这篇网页的价值也就比较高了。这和评价论文的质量一样，论文的被引用次数越多，则说明这篇论文比较重要。像 PageRank[6] 技术就是这类原理。

②URL 本身的质量。通常来说，网站会将最重要的网页放在层次浅的地方，如网站首页以及首页指向的网页都是比较重要的。

③网页重复情况。重复次数过多或者完全不重复的网页，其价值一般都不高。这是因为大部分有价值的网页只会被相关的网页引用有限次。引用过多有作弊嫌疑或者是登录页、注册页等垃圾页面。

④网页内容的评价。如果网页本身存在文字和链接堆砌，正文信息比例太少，标题正文相关性不强，以及出现一些不良特征词等现象，这样的网页的评价质量也不会太高。对于此类网页，需要尽可能准确地识别出来。

4.3 避免下载垃圾的问题

当今的网络已经发生很多变化，我们面临着越来越多的动态网页和垃圾网页。服务端的脚本语言可以产生无穷的主机名和数以百万计的垃圾页面。垃圾主要表现在以下几个方面：(1)下载的网页内容存在大量重复的现象；下载的网页存在隐藏文本、链接的现象；下载的网页存在热门词堆砌的现象；下载的网页内容和锚文本以及标题的相关性不强；(2)下载的网页有价值的内容不多，多数内容是价值不高的锚信息或文字；下载的网页内容和实际用户能访问到的网页内容不一样。SEO(Search Engine Optimization)的出现更加重了这种现象。网络爬行的策略已经从简单的宽度优先搜索(BFS)转为能够实时判定网站是否含有有效内容并给予相应爬行优先级的一种较为智能的方式。文献[9]指出，在爬行了数十亿网页后，BFS 方式会陷入垃圾陷阱中。主要表现为以下几个方面：(a) URL 队列中的垃圾链接已经到了不能忽略的地步，甚至超过了合法的链接；(b) 单个垃圾域会动态地产生大量的主机名，大大降低了 DNS 解析的速度；(c) 网络爬虫大量下载垃圾站点的内容，会造成所有 HTTP 和 DNS 请求的延迟。如何设计出一个低成本、稳健的算法以应对和处理垃圾页面，已成为新一代网络爬虫必须面对的问题。

4.4 网页更新

因为资源的限制，要用重要的网页替换不重要的网页，并同时更新陈旧的信息。更新决定了用户能不能马上搜索到最新的信息，发现最近的热点，屏蔽失效的网页。对整个互联网的数据都重新下一遍的做法显然不可取，这种更新的代价太大，周期也很长，根本达不到更新的目的。在设计 Crawler 时，其更新策略需要满足以下一些要求：在有限的带宽以及时间段内，所有重要网页都需要得到更新；需要确保有效更新的比率；用一套网页更新价值的评估方法，通过网页更新来获得新链接以及网页内容更新带来的索引方面的价值。

目前网页更新的方式有两种[16]：一种是周期性进行更新。Crawler 第一次爬行到设定好的规模后，停止爬行，然后每隔一段(固定的)时间对本地的索引数据库网页进行一次全面的信息更新维护，即替换掉陈旧的信息，加入新生成的网页。更新周期一般以星期或月为计量单位。我们称这种为“周期性 Crawler”。该方法的优点是简单，但由于索引数据库规模巨大，如果在每一次更新过程中，对于每一个 Web 网页都进行有效性验证，其工作量将十分巨大。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。

另一种是增量式信息更新方法。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，只对那些最有可能发生变化的网页进行更新，以不同的频率更新不同的网页。Crawler 会一直不停地爬行，更新陈旧的网页，并用新的“更重要”的网页替换掉“次重要”的网页。我们称采用这种方式的爬虫为“增量式 Crawler”。从理论上来讲，增量式 Crawler 比周期性 Crawler 效率更高。但如何确定每个网页的更新频率，是一个难点。

结束语 网络爬虫技术主要解决的是网络数据的收集问题，这包含数据的覆盖率、信息抽取的准确性、新信息发现的及时性等方面的问题。

在 Crawler 技术的推动下，搜索引擎能将种类更多、质量更好的信息展现给用户。通过技术持续的发展，搜索引擎将不仅是一种提供检索和广告等服务的平台，也将成为一种网络用户多元化沟通和信息共享的平台。

参 考 文 献

[1] Arasu A, Cho J. Searching the Web[J]. ACM Transactions on Internet Technology, 2001, 1(1): 2-43

[2] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[A]// Proceedings of the 6th Conference on Symposium on Opear-ting Systems Design & Implementation[C]. San Francisco, CA, 2004: 10-10

[3] Ghemawat S, Gobioff H, Leung Shun-Tak. The Google File System[A]// Proceedings of the 19th ACM Symposium on Operating Systems Principles[C]. 2003: 20-43

[4] Pike R, Dorward S, Griesemer R. Interpreting the Data: Parallel Analysis with Sawzall [J]. Scientific Programming Journal, 2005, 13: 277-298

[5] Chang F, Dean J, Ghemawat S. Bigtable: A Distributed Storage System for Structured Data[A]// 7th USENIX Symposium on Operating Systems Design and Implementation[C]. 2006: 205-218

[6] Brin S, Page L. The Anatomy of a Large - scale Hypertextual Web Search Engine[J]. Computer Networks, 1998, 30: 107-117

[7] Burner M. Crawling towards Eternity: Building an Archive of the World Wide Web[J]. Web Techniques Magazine, 1997, 2(5): 125-130

[8] Boldi P, Codenotti B, Santini M. UbiCrawler: A Scalable Fully Distributed Web Crawler[J]. Software: Practice & Experience, 2004, 34: 711-726

[9] Lee Hsin - Tsang, Leonard D. IRLbot: Scaling to 6 Billion Pages and Beyond[A]// Proceedings of the 17th International World Wide Web Conference[C]. ACM Press, 2008: 427-436

[10] We knew the web was big [EB/OL]. http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html, 2008-07-25

(下转第 53 页)

参 考 文 献

[1] Idreos S, Tryfonopoulos C, Koubarakis M. Distributed Evaluation of Continuous Equi-join Queries over Large Structured Overlay Networks[A]// Proceeding of the 22th International Conference on Data Engineering[C]. Washington: IEEE Computer Society, 2006: 41-54

[2] Crovella M E, Taqqu M S, Bestavros A. Heavy-tailed Probability Distributions in the World Wide Web. A Practical Guide to Heavy Tails[M]. Chapter 1. New York: Chapman & Hall, 1998: 3-26

[3] Dittrich J P, Seeger B, Taylor D S, et al. Progressive Merge Join: A Generic and Non-blocking Sort-based Join Algorithm[A]// Proceedings of the 28th International Conference on Very Large Data Bases[C]. San Fransisco: Morgan Kaufmann, 2002: 299-310

[4] Farag F, Hammad M A. Adaptive execution of stream window joins in a limited memory environment[A]// 11th International Database Engineering and Applications Symposium[C]. Washington: IEEE Computer Society, 2007: 12-20

[5] Urhan T, Franklin M J. XJoin: Getting fast Answers From Slow and Burst Networks[R]. CS-TR-3994. Computer Science Department, University of Maryland, 1999

[6] Mokbel M F, Lu Ming, Aref W G. Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results[A]// Proceeding of the 20th International Conference on Data Engineering[C]. Washington: IEEE Computer Society, 2004: 251-263

[7] Tao Yufei, Yiu Man Lung, Papadias D, et al. RPJ: producing fast join results on streams through rate-based optimization[A]// Proceedings of the 24th ACM SIGMOD International Conference on Management of Data[C]. New York: ACM Press, 2005: 371-382

[8] Levandoski J J, Khalefa M E, Mokbel M F. Permjoin: An efficient algorithm for producing early results in multijoin query plans[A]// Proceeding of the 24th International Conference on Data Engineering[C]. Washington: IEEE Computer Society, 2008: 1433-1435

[9] 钱江波，徐宏炳，王永利，等. 多数据流滑动窗口并发连接方法[J]. 计算机研究与发展，2005，42(10)：1171-1178

(上接第 29 页)

[11] 中国互联网络发展状况统计报告[EB/OL]. http://www.cnnic.net.cn, 2008-08-01

[12] Cho Junghoo, Garcia-Molina H. Parallel crawlers[A]// Honolulu: Proceedings of the 11th International World Wide Web Conference[C]. ACM Press, 2002: 124-135

[13] 北京大学天网搜索引擎[EB/OL]. http://e.pku.edu.cn, 2005-05-06

[14] Shkapenyuk V, Suel T. Design and implementation of a high performance distributed web crawler[A]// Proceedings of the 18th International Conference on Data Engineering[C]. 2002: 357-368

[15] Najork M, LWiener J. Breadth-first search crawling yields high quality pages[A]// Proceedings of 10th International World Wide Web[C]. ACM Press, 2001: 114-118

[16] Garcia-Molina C. The evolution of the Web and implications for an incremental crawler[A]// Proceedings of the 26th International Conference on Very Large Data Bases[C]. 2000: 200-209

[17] Heydon A, Najork M. Mercator: A scalable, extensible Web crawler[J]. World Wide Web, 1999, 2(4): 219-229

[18] Samaras G, Papapetrou O. Distributed location aware web crawling[A]// Proceedings of the 13th International World Wide Web Conference[C]. ACM Press, 2004: 468-469

[19] Castillo C, Marin M, Rodriguez A, et al. Scheduling algorithms for web crawling[A]// Brazil: WEBMEDIA and LA-WEB[C]. IEEE Cs Press, 2004: 10-17

[20] Boswell D. Distributed High-performance Web crawlers: A survey of the state of the art [EB/OL]. http://www.cs.ucsd.edu/. 2003-12-10

[21] Koht-arsa K, Sanguanpong S. In-memory URL compression[A]// Chiangmai: National Computer Science and Engineering Conference [C]. 2001: 425-428

[22] Najork M, Heydon A. High-performance Web crawling[M]. Handbook of Massive Data Sets. Kluwer Academic Publishers Inc, 2001: 25-45

[23] 李晓明，凤旺森. 两种对 URL 的散列效果很好的函数[J]. 软件学报，2004，15(2)：179-184

[24] Cho J, Garcia-Molina H. Synchronizing a database to improve freshness[A]// Proceedings of 2000 ACM International Conference on Management of Data Conference[C]. 2000: 117-128

[25] Olston C, Pandey S. Recrawl Scheduling Based on Information Longevity [A]// Proceedings of the 17th International World Wide Web Conference[C]. ACM Press, 2008: 437-446

[26] Cai Rui, Yang Jiang-ming, Lai Wei. iRobot: An Intelligent Crawler for Web Forums [A]// Proceedings of the 17th International World Wide Web Conference[C]. ACM Press, 2008: 447-456

[27] Chen Yen-yu, Gan Qingqing. I/O-efficient Techniques for Computing Pagerank[A]// International Conference on Information and Knowledge Management[C]. CIKM Press, 2002: 549-557

[28] 万源，万方，王大震. 一种并行 Crawler 系统中的 URL 分配算法设计[J]. 计算机工程与应用，2006，s1：117-119

[29] 蒋宗礼，赵钦，肖华，等. 高性能并行爬行器[J]. 计算机工程与设计，2006，27(24)：762-765

[30] 张三峰，吴国新. 一种面向动态异构网络的容错非对称 DHT 方法[J]. 计算机研究与发展，2007，44(6)：905-913

[31] 余锦，史树明. 分布式网页排序算法及其传输模式分析[J]. 计算机工程与应用，2004，29：182-186

[32] 沈贺丹，潘亚楠. 关于搜索引擎的研究综述[J]. 计算机技术与发展，2006，16(4)：147-149

[33] 张敏，高剑峰，马少平. 基于链接描述文本及其上下文的 Web 信息检索[J]. 计算机研究与发展，2004，41(1)：221-226

[34] 贺广宜，罗莉. 分布式搜索引擎的设计与实现[J]. 计算机应用，2003，23(5)：83-86

[35] 周雪忠，吴朝晖. 文本知识发现：基于信息抽取的文本挖掘[J]. 计算机科学，2003，30(1)：63-65

[36] 陈华，罗昶，王建勇. 基于 Web 的百万级 FTP 搜索引擎的设计与实现[J]. 计算机应用，2000，20(9)：68-70