第 32 卷 第 20 期 计 算 机 工 程 2006 年 10 月
Vol.32 № 20 Computer Engineering October 2006
·软件技术与数据库· 文章编号：1000—3428(2006)20—0097—03 文献标识码：A 中图分类号：TP311

一种增量式并行 Web 信息采集方法

杨天奇，周 晔
(暨南大学计算机科学系，广州 510632)

摘 要：提出了一个基于多线程并行的增量式 Web 信息采集结构模型，并加以实现，该模型以线程并行的方式对 Web 页面同时采集，实现了全面、高效并且灵活的信息搜集，在系统实现过程中，采取 Java 语言中最新的特性、独特的 URL 调度策略保证了各个线程时间的下载并行与互不相交，页面分析过程为各个线程源源不断地提供下载源，而指纹判别算法保证了并行采集过程中的同步，有效地去除了冗余。对该系统作了测试，实验证明，该系统能有效地提高信息采集性能。

关键词：Web；信息采集；搜索引擎；并行

A Parallel System of Incremental Web Information Gathering

YANG Tianqi, ZHOU Ye
(Department of Computer Science, Jinan University, Guangzhou 510632)

【Abstract】This paper gets into the research on how to crawl information effectively in some sections of Web, which is also called parallel Web crawling technology, and brings forward a structure design model of the parallel incremental Web crawler. In order to download Web pages in parallel, the means of multiple thread and the latest character of Java language are adopted, meanwhile the paper adopts the right means for URL dispatching to make sure that threads would work in parallel with page analysis. In order to reduce redundancy, the method chooses footprint algorithm and extracts URL for threads to download. The test result proves the expect. It can effectively improve information gathering performance.

【Key words】Web; Information gathering; Search engine; Parallel

随着人们对提供的各项信息服务要求越来越高，传统的基于整个 Web 的信息采集也越来越力不从心，它无法及时地采集到足够的 Web 信息。为此，本文展开了对 Web 信息的并行采集研究。在系统实现过程中，采取 Java 语言中最新的特性，独特的 URL 调度策略保证了各个线程时间的下载并行与互不相交，页面分析过程为各个线程源源不断地提供下载源，而指纹判别算法保证了并行采集过程中的同步，有效地去除了冗余。

1 URL 的调度策略

1.1 线程池模型与 URL 队列

根据网络扫描程序的多线程工作及动态分配 URL 等特点，本文设计并实现了一套高效的线程池调度算法和 URL 调度算法。具体算法设计如下：

(1)URL 分配器作为一个独立的线程工作，负责将 URL 链表中的头指针始终指向一个不能放入任何一个 URL 队列中的 URL，当头指针指向的 URL 能放入其中一个队列中时，这个 URL 就将被放入进去(除非这个队列已满就不作处理)，然后再继续指向下一个 URL，直到指向的 URL 已经不能放入任何一个队列中。

(2)为 URL 队列和工作线程各保持一个等待调度的 FIFO 队列，即 URL 队列池和线程池。线程池中按等待时间的先后次序存放空闲的工作线程。URL 队列池按等待时间的先后次序存放无线程处理的 URL 队列。

(3)线程调度器按 FIFO 算法从线程池中取空闲线程，从 URL 队列池中取队列，将其分给空闲线程处理，直到线程池或 URL 队列池为空。

(4)当线程池或 URL 队列池为空时，线程调度器进入睡眠状态。

(5)当工作线程处理完某个 URL 队列后，将自己放回线程池，将 URL 队列放回队列池，若线程调度器为睡眠状态，则唤醒它，线程调度器继续步骤(3)。

(6)当所有线程都空闲，所有 URL 队列都为空，URL 链表也为空时运行结束。

1.2 每个工作线程的抓取流程

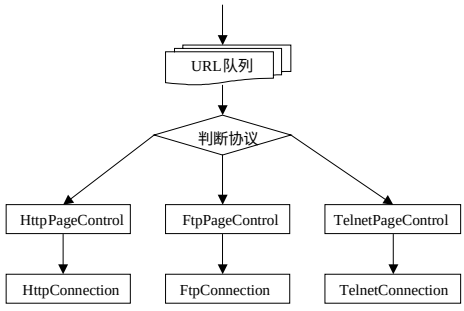

图 1 各工作线程的抓取流程

抓取工作流程如图 1 所示，大致步骤如下：

(1)如果采集数据达到 300kB(上传数据阈值)，则上传数据到信息服务器。

(2)从 URL 队列中取出一个 URL 放入协议判断器中。如果此时 URL 队列为空，则转入步骤(7)，否则转入步骤(3)。

(3)根据 URL 头部的协议信息，协议判断器对 URL 所采用的协

基金项目：广东省自然科学基金资助项目(5006102)
作者简介：杨天奇(1961－)，男，副教授，主研方向：人工智能，数据挖掘，入侵检测；周 晔，硕士生
收稿日期：2005-10-26 **E-mail：**y_tq@163.com

议进行判断，如果是 HTTP 协议，转入步骤(4)，如果是 FTP 协议，转入步骤(5)，如果是 BBS 协议，转入步骤(6)。

(4)启动采集 HTTP 协议的线程 HttpPageControl，对本页通过 HttpConnection 进行采集。采集完后保存此页，转入步骤(1)。

(5)启动采集 FTP 协议的线程 FtpPageControl，对本页通过 FtpConnection 进行采集。采集完后保存此页，转入步骤(1)。

(6)启动采集 Bbs 协议的线程 BbsPageControl，对本页通过 BbsConnection 进行采集。采集完后保存此页，转入步骤(1)。

(7)向信息服务器发送请求 URL 指令，采集完毕。

1.3 页面抓取处理

所抓取的主要页面都是符合 HTTP 协议的，以 HTTP 协议为例。抓取页面的主要实现算法如下：

(1)分析页面 URL，抽出目标站点地址和端口号，若无端口号设为 HTTP 默认端口 80。判断该站点的连接方式设置，若设为直接连接则与该地址和端口建立网络连接；若设为穿越 Proxy 连接则与指定的 Proxy 地址和端口建立网络连接。

(2)若建立网络连接失败，说明该站点不可达，中止抓取该页面并将其抛弃；否则继续下一步骤获取指定页面。

(3)由页面 URL 组装 HTTP 请求头，若该站点需要用户标识和口令则将其填入请求头中，发送请求到目标站点。若超过一定时间未收到应答消息则中止抓取该页面并将其抛弃；否则继续下一步骤分析应答消息。

(4)分析应答头，判断返回的状态码。

(5)若状态码为 2xx，返回正确页面，进入步骤(5)。

(6)若状态码为 301 或 302，表示页面被重定向，从应答头中提取出新的目标 URL，转入步骤(3)。

(7)若状态码为其它，说明页面连接失败，中止抓取该页面并将其抛弃。

(8)从应答头中提取出日期、长度、页面类型等页面信息。若设置了页面抓取限制，进行必要的判断和过滤，抛弃不符合要求的页面。

(9)读取页面内容。对于长度较大的页面，采用分块读取再拼接的方法保证页面内容的完整。至此该页面的抓取完成。

2 页面分析

2.1 HTML 语法分析

在系统中使用的标记文法分析器的基本原理是：由标记文法构造状态转换表，根据输入流中的当前字符(无需向前看)切换状态，当到达特定状态时执行相应语义操作。

2.2 页面中正文的提取

页面的正文提取方法较简单。正文和标记都作为独立的语法成分传递给标记层，标记层根据标记区分出页面标题和内容，并根据系统需要合并出页面的正文。目前还未考虑不同字体或字型的正文的差别。也就是说，在读取页面时，找到标记<body>和</body>，将这两个标记中间的内容去除其中的所有标记即可。

2.3 页面中链接的提取

对一个页面中的链接提取工作流程如下：

(1)从页面文件队列中取出一个页面文件，如果应答头中未说明文件类型，根据 URL 中的文件扩展名补充完整。如果页面文件队列为空，跳转到步骤(7)。

(2)判断页面是否为 text/html/htm/shtml 文件，如果不是，抛弃此文件，转入步骤(1)，否则转入步骤(3)。

(3)从文件头按顺序读取文件，遇到如下标记<a href=…> <area href=…> <base href=…> <frame src=…> <img src=…> <body background=…> <applet code=…>等，记录其中的 URL 连接。如果遇到文件结束符，则跳转到步骤(7)。

(4)将提取出来的 URL 链接按照预先定义的统一的格式补充完整(页面链接中给出的 URL 可以是多种格式的，可能是完整的、包括协议、站点和路径的，也可能是省略了部分内容的，或者是一个相对路径)。

(5)记录下<a href=…> <area href=…> <base href=…> <frame src=…> <img src=…> <body background=…> <applet code=…>等后面对此链接的说明信息。在 URL 与主题的相关性判定中，要用到此信息，并把它定义为扩展元数据。

(6)存储此 URL 及其扩展元数据，跳转到步骤(2)。

(7)页面 URL 提取完毕。

3 指纹判别算法

考虑把这些大量的数据用 16 个 16 进制的数来标记，在数据和标记间建立一一对应的关系，标记后就可以通过仅仅比较这 16 位数的标记来判断是否应该下载网页的内容，大大提高了系统效率。这个标记称为网页或是 URL 的指纹。

3.1 Rabin's fingerprinting 算法及其特性

本系统所使用的指纹生成算法基于由美国哈佛大学教授拉宾(Rabin)提出的方法，其思想如下：

假设$A([b_1,\ldots,b_m])$是包含m个二进制字符的二进制字符串，那么可以根据A构造相应的(m-1)度的多项式如下，其中t是不定元。

$$A(t)=b_1t^{m-1}+b_2t^{m-2}+\ldots+b_{m-1}t+b_m \quad (1)$$

给定一个度为 k 的多项式 P(t)，如下：

$$P(t)=a_1t^k+a_2t^{k-1}+\ldots+a_{k-1}t+a_k \quad (2)$$

那么 A(t) 除以 P(t)的余数 f (t)的度数为(k-1)。对于给定的字符串 A，定义 A 的指纹 f(A)如下：

$$f(A)=A(t) \bmod P(t) \quad (3)$$

拉宾的方案具有如下一些性质：

性质 1 如果字符串 A 的指纹不同于字符串 B 的指纹，那么字符串 A 也不同于字符串 B：

$$f(A)\neq f(B)\Rightarrow A\neq B$$

性质 2 在伽罗瓦(Galois)域：$Z_2=\{0,1\}$上指纹算法满足分配律：

$$f(A+B)=f(A)+f(B) \quad (4)$$

性质 3 计算两个串连字符串的指纹与计算单个字符串得到的指纹和另一个字符串串连后计算得到的指纹相同，即：

$$f(concat(A,B))=f(concat(f(A),B)) \quad (5)$$

性质 4 $p_1t^k \bmod P(t)$等于P(t)去掉首位。

证明：

当 $p_1=0$ 时，$p_1t^k \bmod P(t)=0$；

当 $p_1=1$ 时，$P(t)=p_1t^k+p_2t^{k-1}+\ldots+p_{k-1}t+p_k$；

那么：$p_1t^k \bmod P(t)=-p_2t^{k-1}-\ldots-p_{k-1}t-p_k=P(t)-p_1t^k$。

性质 5 如果 B 的长度为 l，那么：

$$f(concat(A,B))=\{A(t)\,t^l+B(t)\}\bmod P(t)=f(f(A)*f(t^l))+f(B) \quad (6)$$

性质 6 1-位扩展定律

如果$f([a_1,\ldots a_l])=(a_1t^l+a_2t^{l-1}+\ldots+a_{l-1}t+a_l) \bmod P(t)=r_1t^{k-1}+r_2t^{k-2}+\ldots+r_{k-1}t+r_k$

那么$f([a_1,\ldots a_{l+1}])=(a_1t^{l+1}+a_2t^l+\ldots+a_lt+a_{l+1}) \bmod P(t)=(t(a_1t^l+a_2t^{l-1}+\ldots+a_{l-1}t+a_l)) \bmod P(t)+a_{l+1} \bmod P(t)=(r_1t^k+r_2t^{k-1}+\ldots+r_kt+a_{l+1}) \bmod P(t)=(r_2t^{k-1}+\ldots+r_kt+a_{l+1}) \bmod P(t)+r_1t^k \bmod P(t)$ (7)

性质 7 $A(t)+B(t)=A\ XOR\ B$ (8)

3.2 不可约分多项式

在具体实现时，要求多项式 P(t)是不可约分多项式，构造如下：

(1)定义伽罗瓦(Galois)域：$Z_2=\{0,1\}$。

(2)构造$P(t)= a_1t^k+a_2t^{k-1}+\ldots+a_{k-1}t+a_k$ ，其中：

(3)$\begin{cases} a1=1 \\ ai \in Z_2, \forall i \in \{1,...,k\} \end{cases}$

(4)所有的多项式 P(t)都不能包含可以分解的常量项。

(5)对t属于任意的Z_2中的元素，都不能使P(t)的值为 0。

(6)只有通过以上 4 点，P(t)才是不可约分多项式。

因此，对于$P(t)= a_1t^k+a_2t^{k-1}+\ldots+a_{k-1}t+a_k$，要使得P(t)是不可约分多项式必须满足以下条件：

(1)ai ∈Z2, ∀i ∈{1,...,k}

(2)a1 =1

(3)ak =1

(4)$P(t) \neq 0, \forall t \in Z2$ 或 $P(t) \neq 1, \forall t \in Z2$

3.3 算法实现

为实现拉宾算法，现对于一个字节 B，作如下按位拆分，B＝[b1, b2, b3, b4, b5, b6, b7, b8]。

同样，对于一个 32 位的字 W，W=[w1, w2, …, w32]= [W1, W2, W3, W4]；对于一个 64 位的字符串值 D，D＝[d1, d2,…, d64,]= [D1, D2, D3, D4, D5, D6, D7, D8], D[1]= [D1, D2, D3, D4], D[2]=[D5, D6, D7, D8]，如此下去，将每 4 个字节组成一组。

如果 W=[w1, w2, …, w32]= [W1, W2, W3, W4]，则与字符串关联的多项式构造如下：

$$W(t)＝W_1(t)t^{24}+ W_2(t)t^{16}+ W_3(t)t^8+ W_4(t) \quad (9)$$

$W_1(t)＝W_1(t)t^7+ W_2(t)t^6+ W_3(t)t^5+ W_4(t)\ t^4 +W_5(t)t^3+ W_6(t)t^2+ W_7(t)t+ W_8$

$W_2(t)＝W_9(t)t^7+ W_{10}(t)t^6+ W_{11}(t)t^5+ W_{12}(t)\ t^4 +W_{13}(t)t^3+ W_{14}(t)t^2+ W_{15}(t)t+ W_{16}$

$W_3(t) = W_{17}(t)t^7+ W_{18}(t)t^6+ W_{19}(t)t^5+ W_{20}(t)\ t^4 +W_{21}(t)t^3+ W_{22}(t)t^2+ W_{23}(t)t+ W_{24}$

$W_4(t) = W_{25}(t)t^7+ W_{26}(t)t^6+ W_{27}(t)t^5+ W_{28}(t)\ t^4 +W_{29}(t)t^3+ W_{30}(t)t^2+ W_{31}(t)t+ W_{32}$

现令S为待计算指纹的字符串，F_c＝f(S)，W= [W_a, W_2, W_3, W_4]为下一待处理的 32 位的字，则指纹

$$F_n＝f(concat(S，W)) \quad (10)$$

当多项式的度为 32 时，F_c＝[F_{C1}, F_{C2}, F_{C3}, F_{C4}], 由上面的性质可计算如下：

$$\begin{aligned} F_n＝f(concat(S，W))＝f(concat(F_c，W)) \\ ＝(F_c(t)t^{32} + W(t))mod\ P(t) \\ ＝(F_c(t)t^{32})mod\ P(t) + (W(t))mod\ P(t) \quad (11) \\ ＝(F_c(t)t^{56})+(F_c(t)t^{48})+(F_c(t)t^{40})+(F_c(t)t^{32})+(W(t))mod\ P(t) \\ ＝TA(t)+ TB(t)+ TC(t)+ TD(t)+ TW(t) \end{aligned}$$

(此处用TI(t)表示$F_{ci}(t)t^x$)

这样，F_n＝TW *XOR* TA *XOR* TB *XOR* TC *XOR* TD，同样可计算出度为 64、96 等字符串的指纹。

4 算法性能测试效果

在以上理论的基础上，构造并实现指纹生成算法，该算法通过利用 64 度的不可约分多项式对输入的流进行处理，输出 16 位的 16 进制的数，具体实现的流程如下：(1)生成一个度为 64 的不可约分多项式 P；(2)初始化指纹数组 F；(3)将输入流转化为字节数组；(4)将输入的数据每 4 个字节一组与多项式 P 计算，保存结果为 F，最后再计算数据中剩余的 1～3 个字节，将结果与 F 计算得出新的最终结果 F；(5)将 F 转化为 16 个 16 进制的数输出。

表 1 指纹算法性能测试结果

测试次数	处理 URL 条数	CPU 时间(s)	冲突个数	冲突所占比例
1	23 743 961	1 775.43	0	0%
2	23 743 961	1 804.74	0	0%
3	23 743 961	1 846.32	0	0%
4	23 743 961	2 005.60	0	0%
5	23 743 961	1 884.14	0	0%

在一台 CPU 为 Intel PIII 2.8GHz、内存为 256MB、操作系统为 Windows XP 的计算机上进行测试，表 1 是该算法的性能测试结果，该系统能有效地提高信息采集性能。

参考文献

1 Edwards J, McCurley K, Tomlin J. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler[C]. Proceedings of the 10th International World Wide Web Conference, 2001-05: 1245-1249.

2 Keiji Y. A Fast Image-gathering System from the World Wide Web Using a PC Cluster[J]. Image and Vision Computing, 2004, 22(1): 24-28.

3 Lawrence S, Giles C L. Accessibility of Information on the Web[J]. Nature, 2003, 400(6740): 107-109.

4 Merugu, Shashidhar. Adding Structure to Unstructured Peer-to-peer Networks: the Use of Small-world Graphs [J]. Journal of Parallel and Distributed Computing, 2005, 65(2): 54-59.

(上接第 96 页)

```
    strncpy (procs[nprocs].name, obAddr->ImageFileName, 16);
    nprocs++;   }
```

3.2 Windows XP 的检测机制

Windows XP 的检测原理和 Windows 2000 基本一样，但它们的派发机制不同，所以算法也不一样。在 WindowsXP 下主要检测各线程的 WaitListEntry、QueueListEntry 和 ThreadListEntry 所组成的链表。因为在内核中很难定位链表的开始处，所以最好的方法是从一个已知的线程开始根据这个线程的 Tcb 中的链表节点搜索下一个线程的 KTEB，利用 KTEB 得到进程信息(与 Windows 2000 类似)。要找到所有的线程，可以用以下的方式：

```
for each process p on PsActiveProcessList do {
  for each thread t which belongs to process pi do
  findthreadfromlist(t.Tcb.WaitListEntry,t.Tcb.QueueListEntry,t.
Tcb.ThreadListEntry);   }
```

因为要对内核中的数据结构进行操作，所以这些代码都必须运行于内核态，最后将利用上述机制得到的进程信息与利用传统方法得到的进程信息进行比较就可以发现是否存在隐藏的进程了。

4 结束语

判断主机是否已被入侵是计算机安全领域中重要的问题，入侵技术的发展使传统进程检测技术愈来愈不可靠，出现了如 fu rootkit 这样的可以隐藏运行实例的入侵程序。由于这种入侵技术出现得较晚，因此针对隐藏进程展开的各种检测技术也正处于探索中。本文提出了一种基于对 Windows 内核中线程调度链表搜索来检测进程的新思路，它的应用将为传统木马检测工具提供有力的补充。

参考文献

1 武安河, 于洪涛. Windows 2000/XP WDM 设备驱动程序开发[M]. 北京: 电子工业出版社, 2003.

2 Solomon D A, Russinovich M E. Inside Microsoft Windows2000 (Thrid Edition)[M]. 美国: 微软出版社, 2000.

3 Microsoft 公司. Windows Device Driver Development Kit Design Guide[M]. 美国: 微软出版社, 2001.