

基于 Larbin 的网络爬虫体系结构 的研究与改进

李跃健, 朱程荣

(同济大学 计算机科学与技术系, 上海 201804)

摘要: Larbin 是一种开源的网络爬虫/网络蜘蛛, 抓取效率极高。它的 url 去重方法的设计, 效率极高, 占用的内存非常小, 理论上下载 6400 万网页, 使用的内存只有 8M, 然而它的冲突将会对它的性能大打折扣, 实际上当达到 10% 的 url 时就已经有很大的冲突概率, 导致内存利用率的降低以及很多网页不能被抓取。通过研究布隆过滤器, 将 url 的 hash 算法进行改进, 把原本一对一的映射变成多对一的映射, 减小了冲突概率, 同时也将大大地提高 Larbin 在 url 内存方面的利用率。经过实验检验, 使用布隆过滤器, 同样 8M 内存, 当达到 10% 的 url 占有率时, 采用 7 个映射, 可以使得冲突概率最小, 达到 0.82%, 而没采用 Bloom filter 的冲突概率则达到了 10%。

关键词: Larbin; 爬虫; 哈希算法; url 去重; 布隆过滤器

中图分类号: TP309

文献标识码: A

文章编号: 1673-629X(2012)07-0147-04

Study and Improvement on System Architectures of Larbin Web Crawler

LI Yue-jian, ZHU Cheng-rong

(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

Abstract: Larbin is an open source web crawler, it scratches pages very efficiently. On url comparing algorithm, it has great efficiency and cost very little memory. In theory, downloading 64 million pages cost only 8M memory, but its url conflict will greatly affect its performance. In fact, when 10% of the urls are in memory, the new url will have 10% possibility to conflict, resulting in lower memory usage and many pages can not be crawled. By studying the Bloom filter, with the hash algorithm of url distinguish improves the original into a many-to-one mapping, reducing the probability of conflict, and also greatly enhance the Larbin's memory utilization. From the experiment, in the 8M memory with 10% used by url, if make the map number to be 7, the conflict percentage reaches to only 0.82% while it remains 10% if no bloom filter is applied to the algorithm.

Key words: Larbin; web crawler; hash; url distinguish; Bloom filter

0 引言

经过二十多年的发展, 互联网经历了蓬勃的发展, 在这当中最为显著的便是 www, 也即万维网的壮大, 如今已经完全融入到人们日常生活的方方面面。从最初只有几个网站到现今已经超过 1 亿, 而且仍在不断增加当中。网站的丰富给人们带来了更多交流信息的渠道, 同时也提出了一个很大的挑战: 如何能够快速的地找到所需要的有效信息?

为此搜索引擎便应运而生, 从最初的 Archie 到后

来的 Yahoo 再到后来的 Google, 搜索引擎也经历了重大的发展, 特别是到目前由于互联网信息的急剧膨胀, 搜索引擎也开始进行细化分工: 像国外的 Inktomi (已被 Yahoo 收购), 它本身并不是直接面向用户的搜索引擎, 但像包括 Overture (原 GoTo, 已被 Yahoo 收购)、LookSmart、MSN、HotBot 等在内的其他搜索引擎提供全文网页搜索服务, 国内的百度也属于这一类。

一般来讲搜索引擎主要分两部分: 搜和索, 搜指的是爬虫, 索便是收录系统。

爬虫作为搜索引擎的数据采集器, 是搜索引擎的最初数据来源, 对于一个搜索引擎能否很好收录网站有着至关重要的作用。

目前开源的爬虫中, Larbin 无论在性能上还是在稳定性上都出类拔萃, 文中重点研究它的体系结构, 并对其 url 算法去重进行了改进。

收稿日期: 2011-11-30; 修回日期: 2012-03-02

基金项目: 国家 863 高技术发展计划项目 (2010AA122200); 上海市科委国际合作项目 (10510712500)

作者简介: 李跃健 (1985-), 男, 硕士, 研究方向为计算机应用; 朱程荣, 副教授, 研究方向为容错计算及信息安全。

1 Larbin 的工作原理

对于网络而言,基于 TCP/IP 的通信编程^[1]有几种方法:

单线程阻塞:这是最简单也最容易实现的一种,一个例子:在 Shell 中通过 curl 系统命令可以直接实现一个简单的爬虫,然而效率问题也显而易见:由于采用阻塞方式读取,每次的连接伴随着 dns 重新解析,另外在建立连接,写入请求,读取结果这些步骤也会产生一定的延时,从而无法有效地利用服务器的全部资源。

多线程阻塞:建立多个阻塞的线程,分别请求不同的 url。相对于单线程阻塞,它能更有效地利用机器资源,特别是网络资源。无数线程在同时工作,所以网络能够比较充分的利用,但无数的线程会很快把 CPU 资源消耗,线程的频繁切换也会带来性能上的损失。

单线程非阻塞:这是目前使用的比较多的一种做法,无论在客户端还是服务器都有着广泛的应用。在一个线程内打开多个非阻塞连接,通过 poll/epoll/select 对连接状态进行判断,在第一时间响应请求,不但充分利用了网络资源,同时也将本机 CPU 资源的消耗降至最低。这种方法需要对 dns 请求,连接,读写操作都采用异步非阻塞操作,它一方面能够建立尽量多的连接,另一方面又不会占用大量的服务器资源,特别是 CPU 资源,Larbin 就是采用了这种编程模式。

2 Larbin 的模块结构

图 1 为 Larbin 体系各模块关系^[2]。

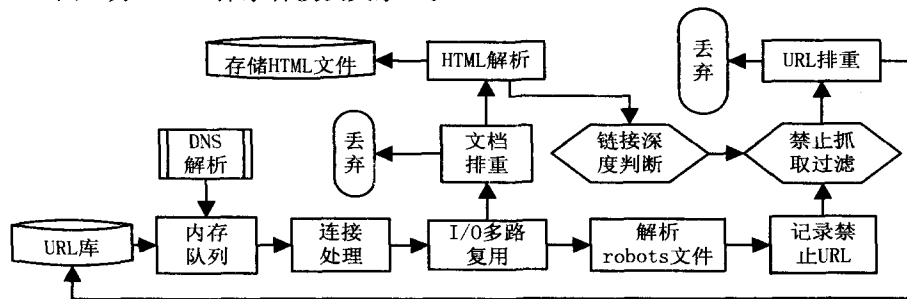


图 1 Larbin 体系各模块关系

Larbin 采用了模块化的程序设计,系统分为以下几大模块^[3]:url 模块、文件操作模块、DNS 模块、html 解析模块、后台读取链接服务器模块、后台统计信息服务器模块。

下面通过 Larbin 服务器运行的一整个流程来剖析各模块的运作情况:

服务器启动后,调用 glob() 函数进行初始化:首先建立系统需要用的各全局变量,然后打开 larbin.conf,读取相应的参数,传给相应的全局变量。比较重要的有 URLsDisk、URLsDiskWait,类型为 PersistentFifo,主要管理 url 的调度,包括 url 的插入、校验,从磁盘读取

url,将 url 写入磁盘等功能。它采用了多级队列的方式,有效地实现了缓存,当内存中的 url 超过一定的数量时便将队列写入磁盘,当需要解析时再读取回来,保证速度与内存开销的平衡;另外还初始化了 named-SiteList 数组,用来存放解析的 dns^[4]。前面已经说过,Larbin 的 DNS 模块是独立出来的,这样可以充分降低解析次数,理论上只要解析过一次就不需要再进行解析,大大提高了效率。

接下来就是服务器开启 startThread(startWebserver, NULL) 进程^[5],启动微型服务器,便于用户查看 Larbin 的运行状态,它能每过一段时间便刷新状态,所以客户端打开浏览时都能查看到即时的情况。

然后服务器开始进入 URL 队列调度和处理过程,队列调度由 cron() 函数执行

```
if ((global::now % 300) == 0) {
    global::readPriorityWait = global::URLsPriorityWait->getLength();
}
global::readWait = global::URLsDiskWait->getLength();
}
if ((global::now % 300) == 150) {
    global::readPriorityWait = 0;
    global::readWait = 0;
}
```

这段代码是 url 调度的核心代码,调度的过程如图 2 所示^[6]:global::now % 300 是判断这次是对 wait 里的 url 进行处理,还是对不是 wait 里的 url 进行处理。% 300 为 0 或者 150 的概率都是 1/300,所以大约 300 次

换一次。处理时,将 url 加入队列 NamedSiteList,同时更新这个主机的 DNS,调用 transfer() 函数将解析的 IP 地址存入 IPSiteList 队列^[7]。

```
/* Get the next url
 * here is defined how priorities
 * are handled
```

```
*/
static bool canGetUrl(bool * testPriority) {
    url * u;
    if (global::readPriorityWait) {
        global::readPriorityWait--;
        u = global::URLsPriorityWait->get();
        global::namedSiteList[u->hostHashCode()].putPriorityUrlWait(u);
        return true;
    } else if (* testPriority && (u = global::URLsPriority->tryGet()) != NULL) {
        // We've got one url (priority)
        global::namedSiteList[u->hostHashCode()].putPriorityUrl
```

```

(u);
return true;
} else {
    * testPriority = false;
    // Try to get an ordinary url
    if ( global::readWait ) {
        global::readWait--;
        u = global::URLsDiskWait->get();
        global::namedSiteList[ u->hostHashCode() ]. putUrlWait
    }
    (u);
    return true;
} else {
    u = global::URLsDisk->tryGet();
    if ( u != NULL ) {
        global::namedSiteList[ u->hostHashCode() ]. putUrl(u);
        return true;
    } else {
        return false;
    }
}
}
}
}
}

```

URL的调度

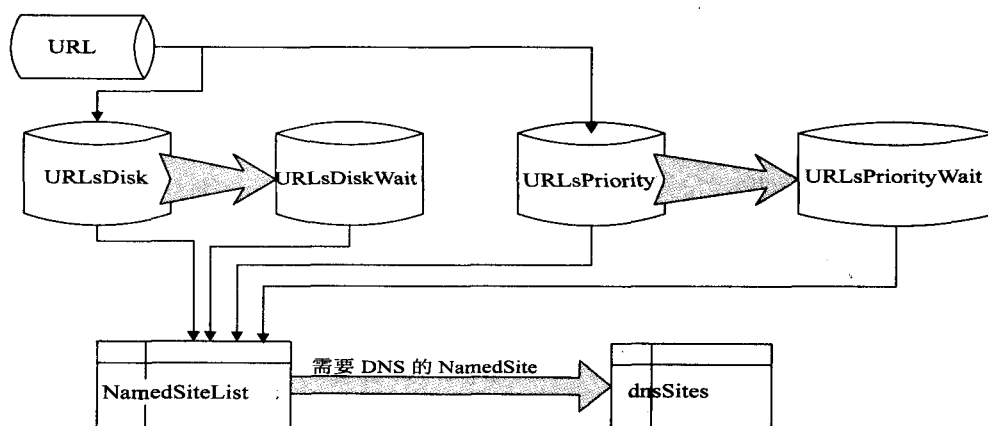


图2 url的调度

接下来 main() 函数中开始执行 fetchDns(), 从 dnsSites 中取得解析测试的站点名, 然后用 newQuery() 提出解析请求, 当检测状态通过时, 调用 dnsAns(), 更新 DNS 列表。

下一个调用的是 fetchOpen(), 得到站点后调用 fetch(), 对将 url 指向的文件下载下来, 并对 html 代码进行解析, 提取其中的 url 地址加入队列^[8]。

接下来又开始重复执行以上各步骤, 不段地爬下去。

3 Larbin 的特点

3.1 url 处理

经过原代码分析与实际测试可以发现, 由于

Larbin 对 url 队列的去重处理已经在各队列索引当中普遍采用了 hash 算法, 所以运行效率非常高。另外由于采用了单线程非阻塞的模式, 并发的线程可以达到很快的下载速度, 在带宽允许的条件下, 一天就能下载 500G 的网页, 但同时也正由于它的 hash 算法没有对冲突进行解决, 只是采取了简单的无视, 直接导致了一些 url 不被抓取, 而且特别是当队列里的数量更大时, 冲突的概率更大^[9]。

下面来看看它所采用的 url 去重算法:

Larbin 中 hash 表的定义是先建立一个数组, 长度为 hashSize/8, 其中 hashSize 定义在 type.h 中, 大小为 64000000, 所以根据定义, 首先会申请大约 8M 的空间。这是 hashTable 的 hashCode 的算法:

```

uint url::hashCode() {
    unsigned int h = port;
    unsigned int i = 0;
    while ( host[i] != 0 ) {
        h = 31 * h + host[i];
        i++;
    }
}

```

```

i = 0;
while ( file[i] !=
0 ) {
    h = 31 * h + file
[i];
    i++;
}
return h % hash-
Size;
}

```

根据推算, 64000000 个 bit 需要 8M 的空间, 理论上可以存放大约

6400 万个 url, 然而假如算起来一个网站有 1000 个 url 的话, 这样理论上来说就只能存放 6.4 万个网站, 然而伴随着 url 的增多会导致冲突概率的增大, 当数据达到 6000 个网站的时候冲突概率便能达到 10%, 而且会随着 url 增多冲突可能性急剧恶化。

3.2 dns 解析

同一站点内 dns 请求可以只做一次, 可以采用 dns 缓冲的方式将主机名独立于 url, 解析完成的 IP 存入队列, 用于发起连接使用。同样, Larbin 在存储时也采用了 hash 码的映射存储, 优点是速度快, 缺点也与上面的情况类似。在 type.h 里定义了 namedSiteListSize 为 2 万, 也就是说当达到 2000 个网站时冲突概率便会有 10%, 所以从这个角度上说 Larbin 会产生误判的概率很大的。

4 改进方案

可以看出, Larbin 源代码当中多处涉及到 hash 算法^[10], 对于一个企业级搜索引擎来说, 如何更高效地提高它的效率与正确性呢? 这当中改进 hash 算法就是一个比较好的选择, 它能在多方面提高它的整体性能, 包括: url 去重、队列赋值、dns 解析等等。

哈希算法是将任意长度的二进制值映射为固定长度的较小二进制值, 这个小的二进制值称为哈希值, 然而接下来会遇到比较大的问题: 如何解决冲突, 接下来发现很多人把绝大的精力花在了如何处理冲突这个问题上, 问题是解决了, 但是性能也会受到非常大的影响, 而如何针对实际应用绕开冲突倒不失为一个很好的方法。

对于搜索引擎而言, 一定数量的误判是可以忍受的, 譬如说 1%, 而如何按照传统的哈希算法, 即使哈希函数足够精妙, 设想要抓取一万个网站, 每个网站平均 1000 个网页, 还是需要 $1 * 10000 * 1000 / 1\% / 8 = 125\text{M}$ 左右的空间, 这还是保守估计, 所以如何有效地利用空间同时保证较低的误判? 经过研究发现, 可以采用多个哈希码映射, 这样能够大大降低冲突的概率。

如图 3 所示^[11], 对地址进行多个哈希运算, 计算出四位地址用来存储哈希, 这样可以大大减少冲突的概率。

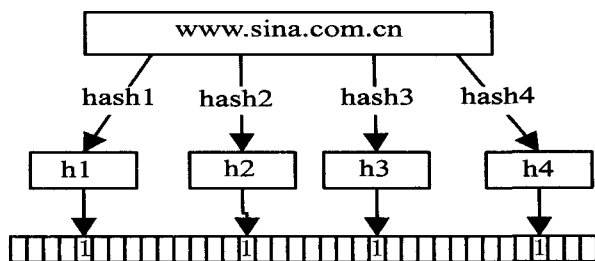


图 3 将 url 映射到向量空间

5 改进方案的优势

这种方法继承了本身哈希表的高效性, 包括空间上采用位运算、查找时间为线性复杂度等等, 另外也避免了处理冲突所带来的异构及维护的复杂性^[12]。

根据这个算法^[13], 对于有 m 个 bit 位的哈希表, 如果想要保证 1% 的误判率, 则这个 bit array 只能存储 $m/100$ 个元素, 因而有大量的空间被浪费, 而采用了 K 个 hash 映射, 即 k 个 hash function 将每个元素改为对应于 k 个 bits, 因为误判度会降低很多, 并且如果参数 k 和 m 选取得好, 一半的 m 可被置为 1, 举例来说如果 m 值为 1 亿, 传统的哈希算法会处理冲突, 时间复杂度为:

$$\frac{1}{1 - \frac{n}{m}}$$

如果哈希表满半 (即 $n/m = 1/2$), 每次搜索需要探测 2 次, 每个元素占 8 位, 总空间是

$$\frac{10^8 * 8 \text{ bytes}}{50\%} = 1.6 \text{ GB}$$

如果采用新的算法, 同样的 n/m 达到 1/2, 因为采用的是位操作, 所以只有 1/8 的空间, 为 200M, 所需的空間小很多, 而且误判的概率为百万分之一。误判的概率为 (K 指映射的数量)^[14]:

$$f(x) = (1 - e^{-\frac{nk}{m}})^k$$

通过运算^[15], 当 $K = P(\text{error}) = (1 - \frac{1}{2})^k = 2^{-k} =$

$2^{-\ln_2 \frac{m}{n}} \approx 0.6185^{\frac{m}{n}}$ 时误判率最低, 也就是说 m/n 值保持线性就能保证误判率的稳定。

6 结束语

文中深入研究了 Larbin 这个开源网络爬虫的源代码, 分析了其体系架构, 深入探讨了它的一些核心算法, 总结了它在爬虫算法上的一些优点, 同时也发现了它在 url 去重算法的不足, 于是研究了一套多维哈希去重算法以弥补原先算法高内存消耗的不足。经过论证采用新的算法后可以大大地提高 url 队列的内存利用率并减小误判率。

参考文献:

- [1] 崔滨, 万旺根, 余小清, 等. 基于 EPOLL 机制的 LINUX 网络游戏服务器实现方法[J]. 微计算机信息, 2006, 22(21): 64-66.
- [2] 孟时, 王彦. larbin 网络爬虫的体系结构[J]. 电脑学习, 2010(8): 80-81.
- [3] 王锋, 王伟, 张璟, 等. 基于 Linux 的网络爬虫系统[J]. 计算机工程, 2010(1): 280-282.
- [4] 李刚, 周立柱, 郭奇, 等. 领域相关的 Web 网站抓取方法[J]. 计算机科学, 2007(2): 137-140.
- [5] 王小林, 刘宏申. 搜索引擎的设计研究[J]. 计算机技术与发展, 2007, 17(2): 6-7.
- [6] 王芳, 陈海建. 深入解析 Web 主题爬虫的关键性原理[J]. 微型电脑应用, 2011, 27(7): 32-34.
- [7] 袁浩, 黄烟波. 网页标题分析对主题爬虫的改进[J]. 计算机技术与发展, 2009, 19(6): 22-28.
- [8] 薛峰, 赵问道. 基于最大网络收益的 DNS 内容路由算法[J]. 浙江大学学报, 2004, 38(10): 1270-1273.
- [9] 张帆, 李琳娜, 杨炳儒. 基于 Web 的智能信息采集及处理系统设计与实现[J]. 计算机工程, 2007, 33(18): 265-267.
- [10] 王后珍, 张焕国, 杨飏. 多变元 Hash 函数的构造与分析[J]. 电子学报, 2011, 39(1): 237-241.
- [11] 丁振国, 吴宝贵, 辛友强. 基于 Bloom Filter 的大规模网页

令,只要向相应的主节点提出即可^[11,12],如图 4 所示。

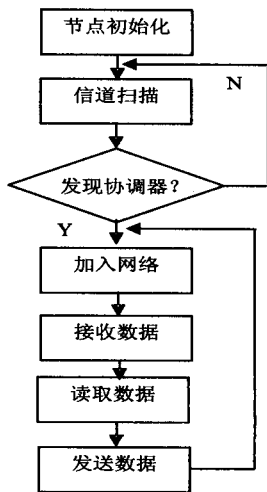


图 4 终端节点程序流程图

3 系统的测试结果

将上述系统配置好后,使用 CC2430 和 SHT11 构建的温湿度监测系统就可以正常运行了,将主节点连接到 PC 机的 RS232 串口,打开串口调试软件或者超级终端,就可以获得主节点汇聚的来自子节点的温湿度数据,同时还可以获得各个子节点的唯一 16 位短地址,用来区分各个子节点。在实验室内部署了 6 个 ZigBee 无线传感模块,包括 1 个主节点,5 个子节点。某一时刻获得的温湿度信息数据如表 1 所示:

表 1 实验室某时刻测得的温湿度信息

短地址	温度值	湿度值
0x1	18℃	48
0x2	21℃	46
0x3	18℃	46
0x4	18℃	48
0x5	18℃	47
0x6	19℃	46

4 结束语

在文中,设计和实现了基于 ZigBee 技术的无线温湿度传感器网络监测系统。采用星型网络结构,集成了 8051 内核的无线射频芯片 CC2430 以及高集成度的数字温湿度传感器 SHT11 构建硬件节点,设计并实现了节点软件。通过在实验室的实际测试,表明本系统具有低功耗、低成本、易于维护扩展等优点,具有很好的实用性。是现有有线温湿度监测系统的理想替代设计。

参考文献:

- [1] 王 殊,阎毓杰,胡富平,等. 无线传感器网络的理论及应用[M]. 北京:北京航空航天大学出版社,2007:6-10.
- [2] CHIPCON. CC2430 PRELIMINARY data sheet (rev. 1.03) SWRS036A[M]. [s.l.]:CHIPCON,2005.
- [3] Mhatre V, Rosenberg C. Design Guidelines for Wireless Sensor Networks Communication, Clustering and Aggregation[J]. Ad Hoc Networks Journal,2004(2):45-63.
- [4] Hill J L. System Architecture for Wireless Sensor Networks [D]. Berkeley:UCA,2003.
- [5] 张艳丽,杨仁弟. 数字温湿度传感器 SHT11 及其应用[J]. 工矿自动化,2007(3):113-114.
- [6] 杜鹏雷,吴 晓,杨丽平,等. 面向精准农业的感知节点传感器驱动与控制[J]. 计算机技术与发展,2010,20(6):233-236.
- [7] 王 戈,张效义. 可用于环境监测的无线传感器网络节点的设计[J]. 传感器与微系统,2007,26(10):117-120.
- [8] 刘子京,裴文江. 基于 ZigBee 协议的无线传感器网络研究[J]. 计算机技术与发展,2009,19(5):192-194.
- [9] 敬海霞,胡向东. 无线传感器网络的路由协议研究[J]. 计算机技术与发展,2007,17(10):150-154.
- [10] 杨永雷,朱 军. 无线传感器网络中异步成簇算法的研究[J]. 计算机技术与发展,2010,20(2):145-151.
- [11] 饶云华,代 莉. 基于无线传感器网络的环境监测系统[J]. 武汉大学学报,2006,52(3):345-348.
- [12] 唐旋来,汪秉文,汤 强,等. 无线传感器网络在桥梁健康监测中的应用研究[J]. 计算机技术与发展,2011,21(1):121-125.

(上接第 150 页)

- 去重策略研究[J]. 现代图书情报技术,2008(3):45-50.
- [12] 吴 军. 布隆过滤器——一种新的海量信息的 hash 算法[R]. [s.l.]:google 研究院,2008.
 - [13] Broder A, Mitzenmacher M. Network applications of bloom filters;a survey[J]. Internet Mathematics, 2005, 1(4):485-509.
 - [14] Mitzenmacher M. Compressed Bloom Filters[J]. IEEE/ACM Transactions on Networking,2002,10(5):604-612.
 - [15] Mitzenmacher M. A second look at bloom filters[J]. Communications of the ACM,1983,26(8):570-571.