第 35 卷 第 2 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.2 Computer Engineering January 2009
·软件技术与数据库· 文章编号：1000—3428(2009)02—0034—03 文献标识码：A 中图分类号：TP311

基于流水线负载平衡模型的并行爬虫研究

孟祥乾，叶允明，邓 斌
(哈尔滨工业大学深圳研究生院，深圳 518055)

摘 要：针对并行爬虫系统在多任务并发执行时所遇到的模块间负载平衡问题，提出流水线负载平衡模型(PLB)，将不同的任务抽象为独立模块而达到各模块的处理速度相等，采用多线程的方式实现基于 PLB 的并行爬虫，根据线程的休眠和缓冲区的变化对线程数量进行动态调整以实现 PLB。实验结果表明该方法具有良好的运行效率和稳定性。

关键词：爬虫；并行；流水线；负载平衡

Study on Parallel Crawler Based on Pipeline Load Balancing Model

MENG Xiang-qian, YE Yun-ming, DENG Bin
(Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen 518055)

【Abstract】 This paper proposes a load balancing model named Pipeline Load Balancing(PLB), to address the load balancing problem among concurrent modules in a parallel crawling system. Different tasks in PLB are implemented as independent modules which have similar processing abilities. Dynamic multi-threading and buffering mechanisms are employed to implement a PLB-based parallel crawler. The number of threads is adjusted according to the changing in buffer size and waiting interval of a thread. Experimental results show that the PLB-based crawler provides high performance as well as good stability.

【Key words】 crawler; parallel; pipeline; load balancing

高效的网络爬虫(crawler)是搜索引擎的重要基础。采用多任务并发执行，实现类似于CPU的流水线(pipeline)运行方式，可极大地提高网络和计算资源的利用率等性能。流水线稳定高效运行的关键在于各执行单元间的动态负载平衡，但目前对并行爬虫的研究大多集中于对任务进行有效分割和各模块的实现方法上[1-4]，很少涉及多任务间的并发执行。本文研究了并行化爬虫中流水线式的动态负载平衡问题，抽象出流水线负载平衡(Pipeline Load Balancing, PLB)模型，介绍了针对该目标模型的多线程式实现方法，包括增加与减少线程数量的测算与实现算法，通过对比实验验证了该模型及其实现的有效性。

1 面向网络爬虫的 PLB 模型

一个爬行对象的爬行过程需执行如下的任务[5]：对一个URL进行DNS，获取主机对应的IP地址，再通过HTTP协议下载该页面，从中抽取超链接，并根据所定义的过滤规则逐个进行过滤，最后将其作为新的爬行对象加入待爬行队列中。采用单任务串行的方式执行，即任意时刻只有一个任务在运行，必然会导致网络和计算资源的浪费。

本文提出采用爬虫的流水线运行方式，即多个任务同时并发运行。参考多种爬虫的设计，多任务并行化可设计为如图 1 所示的结构，将各个任务设计成相对独立的模块，利用缓冲区进行连接，前面的模块在处理完一个爬行对象后将其放入缓冲区中，再由后面的处理模块从缓冲区中取出并处理。采用这种实现方式，如果没有模块间的负载平衡，则会出现一系列问题：处理速度快的模块运行完成后会等待处理速度慢的模块，从而导致计算资源的浪费；而较慢模块的缓冲中需要保存大量的待处理文件，容易导致内存溢出。因为爬虫系统中各种任务的处理速度并不固定，受到网络环境、内存大小等因素的影响波动较大，所以实现动态的负载平衡调整十分必要。

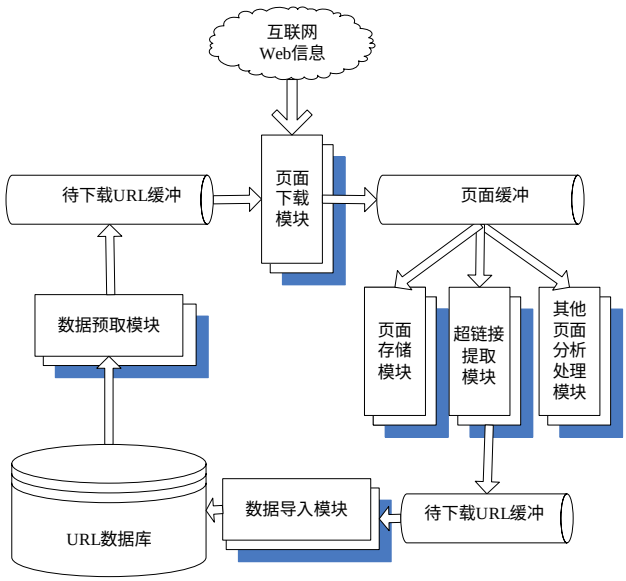

图 1 爬虫系统结构与工作流程

对 PLB 的抽象建模是以图 1 中的实现方式为基础，为了描述并行爬虫的 PLB 模型，定义如下的符号：

M_i：以处理过程为顺序的第i个模块，例如模块M_2要处理

基金项目：国家自然科学基金资助项目“基于增量学习的主题爬虫关键技术研究”(60603066)

作者简介：孟祥乾(1983－)，男，硕士研究生，主研方向：网络爬虫，信息检索；叶允明，副教授、博士；邓 斌，硕士研究生

收稿日期：2008-06-17 **E-mail：**worldmxq@gmail.com

的是模块M_1的结果；

$p_i(j)$：在时间段j内，第i个模块所处理的爬行对象总数。

流水线稳定高效运行的关键在于各执行单元的处理能力相匹配，即各模块的处理速度均衡，满足

$$p_1(j)=p_2(j)=\cdots=p_n(j)=c \tag{1}$$

其中，存在如下 2 个问题：

(1)如何确定常数 c

常数 c 的实际意义为单位时间内系统任意模块所处理爬行对象的数量，在该模型下也可看作整个流水线的处理速度。在实际中存在多种因素影响系统处理能力，包括用户的需求、网络环境、计算资源等。对常数 c 的控制将直接影响爬虫的整体表现，因此，是对爬虫性能进行精确控制的良好方法。

(2)如何调整M_i的计算资源，保持等式成立

PLB 的目标是使每个模块的计算资源与其处理能力相匹配，没有模块因为速度过慢而导致缓冲区过分膨胀，也没有模块因为速度过快而导致计算资源浪费。

2 基于 PLB 模型的并行爬虫实现

本文采用多线程方式实现图 1 所示的处理框架，将模块实现为线程组，由若干执行该任务的线程以及对该组线程进行监控的 controller 组成，controller 通过增加或减少线程的数量实现系统对计算资源的分配。

2.1 多线程方式的 PLB 目标模型

在不失一般性的条件下，为便于进行抽象，需要作出如下假设：

(1)不考虑缓冲区中加入或者减少一个爬行对象的互斥时间，即一个爬行对象的加入或删除是瞬时完成的。

(2)线程不会异常退出，即不会出现该线程未退出但不能处理爬行对象的现象。

(3)同一模块中的任意线程对同一爬行对象的处理速度是相同的。

同时定义如下的符号：

$N_i(j)$：表示第i个模块在时间段j的线程数量。

B_i：连接模块i-1 和模块i的缓冲。模块i-1 将处理结果写入B_i，而模块i从B_i中获取需要处理的爬行对象。

B_i：缓冲区B_i的大小，$B_i(j)$表示在时间段j结束点上的缓冲区大小。

v_i：第i个模块的一个线程对爬行对象的处理速度，$v_i(j)$表示在时间段j内的平均处理速度。

Δt：一个时间段内的时长。

$p_i(j)$：在时间段 j 内，第 i 个模块所处理的爬行对象总数。在模块 i 所有线程均满负载运行的情况下有

$$p_i(j)=n_i(j)\cdot v_i(j)\cdot\Delta t$$

根据之前的讨论，流水线达到负载平衡时，同一时间段内各模块处理的爬行对象个数相同，且又达到每个线程的满负载运行，没有出现空等。因此，将 $p_i(j)=n_i(j)\cdot v_i(j)\cdot\Delta t$ 代入式(1)中得到

$$n_i(j)\cdot v_i(j)=c' \tag{2}$$

其中，c' 为常数。

这样便得到了多线程实现 PLB 的目标模型。即每个模块的线程数量与该模块中一个线程对爬行对象的处理速度成反比。

在实现中，根据时间段 t 内下载速度的变化计算得到下一时间段(t+1)内的线程数量，力争使(t+1)时间段内的线程数量与处理速度相匹配，但是这时并不知道(t+1)时间段内的处理速度，所以这里将其预测为(t)时间段内的处理速度，而在(t+2)时间段内线程数量的调整以(t+1)时间段内预测的处理速度与实际处理速度之差为依据，即假设同一模块中线程在一个时间段内的处理速度与上一时间段内相同。

$$V_i(t+1)=V_i(t) \tag{3}$$

下面对线程数量的调整策略均以该假设为基础。

2.2 基于线程休眠的线程数量控制策略

如果在时间段t内模块M_2的处理速度大于M_1，但实际上两者处理总量相同。原因在于M_2中的部分线程由于没有可供其处理的下载对象而空等，如果将M_2中所有线程空等的时间与若干线程的运行时间交换，则可以认为这些线程在t内一直处于空等状态，因此，这些线程的数量即为M_2中应减少的数量。

设在时间段t内，M_2模块中所有线程的休眠时间总和为S，则$\lfloor S/\Delta t \rfloor$就可看作$M_2$中一直处于空等的线程数量。则在$t$+1时间段内$M_2$模块应减少：

$$\Delta n_{2-}=\lfloor S/\Delta t \rfloor \tag{4}$$

2.3 基于缓冲区大小的线程数量控制策略

当t时间段内M_2的整体处理能力小于M_1时，M_2将不能及时处理M_1所产生的结果，会出现连接这 2 个模块的缓冲区B_2的数量b_2增长，因此，需要增加n_2以增加M_2的处理能力。

在t时间段内，M_2所不能及时处理的爬行对象数量($p_1(t)$-$p_2(t)$)将全部进入缓冲区B_2中，即t时间段内b_2的增加量为($p_1(t)$-$p_2(t)$)，即

$$\Delta b_2(t)=b_2(t)-b_2(t-1)=p_1(t)-p_2(t)$$

所以，调整的目标是达到

$$n_2(t+1)\cdot v_2(t+1)\cdot\Delta t=n_1(t+1)\cdot v_1(t+1)\cdot\Delta t \tag{5}$$

根据假设，n_1, v_1, v_2都不会变，因此有

$$n_2(t+1)v_2(t)\cdot\Delta t=n_1(t)\cdot v_1(t)\cdot\Delta t=p_1(t)=$$
$$p_2(t)+\Delta b_2(t)=n_2(t)\cdot v_2(t)\cdot\Delta t+\Delta b_2(t)$$

可以得到

$$n_2(t+1)=n_2(t)+\frac{\Delta b_2(t)}{v_2(t)\cdot\Delta t} \tag{6}$$

得到M_2应增加的线程数量为

$$\Delta n_{2+}=n_2(t+1)-n_2(t)=\frac{\Delta b_2(t)}{v_2(t)\cdot\Delta t} \tag{7}$$

但是这里 Δt 不属于已知量，应该用别的变量代替，将 $p_1(t)=n_1(t)\cdot v_1(t)\cdot\Delta t$ 代入后得到

$$\Delta n_2=\frac{\Delta b_2(t)}{p_1(t)}n_1(t) \tag{8}$$

该结果存在一个缺点：增加数量为 Δn_{2+} 的线程只能使M_2的处理能力与M_1匹配，而不能将之前缓冲区内的数据增加量 $\Delta b_2(t)$ 消除，如果存在多次处理速度不匹配，必将导致系统内存的溢出。所以，必须对 $\Delta b_2(t)$ 进行消除操作，使一个或者若干时间段内M_2的处理速度略大于M_1，将之前增加的数据逐渐减少。这里假设系统设定在n时间段内将 $\Delta b_2(t)$ 消除，且 2 个模块的处理能力在这段时间内不会产生变化，那么有

$$n_2(t+1)\cdot v_2(t)\cdot\Delta t=n_2(t)\cdot v_2(t)\cdot\Delta t+\Delta b_2(t)+\frac{1}{n}\Delta b_2(t) \tag{9}$$

即每个时间段内M_2多处理了 $\frac{1}{n}\Delta b_2(t)$ 个爬行对象，对式(5)按照上面的方法求解，得

$$\Delta n_2=\frac{(n+1)\Delta b_2(t)}{np_1(t)}n_1(t) \tag{10}$$

需要说明的是，该结果在下面 n 时间段内 2 个模块处理速度不发生变化的情况下有效，而这在实际运行环境中是很难达到的。因此，这个结果仅提供了一种使线程数量与处理速度相融合的趋势，不可能达到完全精确的匹配。

这里以缓冲区数量的变化为参数对线程数量进行调整，原因在于其能够直接反映缓冲区数量的变化，防止溢出，且缓冲区大小的测算较易实现。

3 实验分析

实验中运行了 3 组测试，均以总共 10 个国内大型门户网站和知名高校网站为种子站点开始爬行。第 1 组运行实现了并行化 PLB 的系统，爬行和 URL 抽取实现为独立的模块，并动态调整这 2 个模块的线程数量；第 2 组运行串行化多线程系统，每个线程都完成下载、分析和数据导入，线程数量是第 1 组测试中 2 种线程平均量之和；第 3 组测试为并行化但没有 PLB，线程数量为第 1 组测试中的爬行线程的平均值。

每组测试运行时间为 7 h，每 10 min 记录一次该时间段内网络带宽占用的平均值。这 3 组测试的下载速度的趋势比较见图 2，下载总量的比较如表 1 所示。

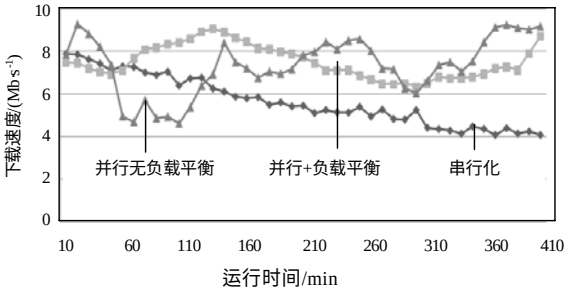

图 2　下载速度趋势

表 1　下载总量比较

测试组	下载总量/GB
并行+负载平衡	16.18
并行无负载平衡	15.07
串行化	12.16

从图 2 中可以看出，串行化的系统下载速度呈下降趋势，这是因为在对数据库操作时，数据量的变大造成数据库操作所占用时间变长。并行化系统中则不存在这个问题，虽然下载速度不是很稳定，但整体维持在一个较高的水平上，下载速度快于串行化的系统，而且这个差别会越来越大。同时未实现负载平衡的并行系统中下载速度的变化较为剧烈，实现负载平衡后，系统可以较好地平滑网络速度的变化。而从表 1 中可以看出，未实现负载平衡的下载总量也要少于实现了负责平衡的系统，原因可以从表 2 所示的 2 组运行参数比较中看出。

表 2　实验运行时参数比较

测试组	缓冲区平均数量	休眠线程数量
并行+负载平衡	167	17
并行无负载平衡	294	35

缓冲区是指爬行线程将页面下载后放入的存储空间，URL 抽取模块的线程从该缓冲区中提取数据。该缓冲区中的元素是所下载网页，平均一个元素约占用 20 KB 的空间，因此该缓冲区对系统内存消耗有直接关系，系统中对过滤模块 URL 进行过滤的能力也是由剩余内存数量确定的，在内存减小的情况下，该数量会降低，这样会导致数据库操作的负担加大，下载性能相应地降低。同样，处于休眠状态的线程数量越多，处于工作状态的线程数量就越少。因为处于休眠状态的线程也会占据一定的计算资源，所以休眠线程的增多也会降低系统性能。

在线程数量的调整策略中，本文采用了在下一个时间段内消除上一时间段内的缓冲区增量。从图 3 中可以看出，线程数量变化较为剧烈，这是因为缓冲区数量的增加导致线程数量增加了 2 倍，而线程数量增加后会造成缓冲区数量的相应减少。这样便形成了图 3 所示的线程数量变化和缓冲区数量变化呈相反的趋势。这些实验结果表明，基于 PLB 模型的爬虫除了具有高性能的特点外，对缓冲区的利用效率也比较高，而且稳定性好。

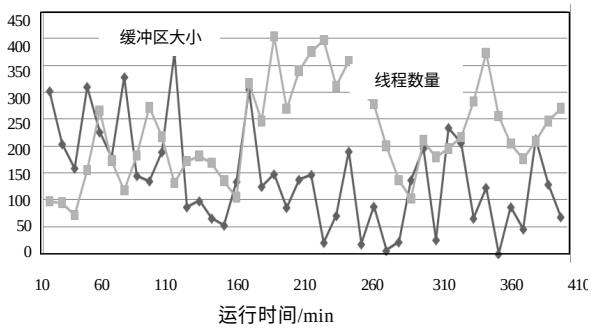

图 3　线程数量变化与缓冲数量变化对比

4 结束语

本文主要研究了并行爬虫系统中的关键问题——模块间负载平衡问题。针对网络爬虫的运行特点，提出了流水线负载平衡模型，并采用多线程和缓冲区调度方法实现了多任务的并发执行及流水线式运行方式。通过研究针对多线程的 PLB 目标模型，得到了通过对线程数量以及缓冲区大小进行动态调整来达到平衡的方法。最后通过综合的实验对比，验证了该方法具有良好的运行效率和稳定性。

参考文献

[1] Brin S, Page L. The Anatomy of a Large-scale Hypertextual Web Search Engine[C]//Proc. of the 7th International Conference on World Wide Web. [S. l.]: IEEE Press, 1998.

[2] Shkapenyuk V, Suel T. Design and Implementation of a High Performance Distributed Web Crawler[C]//Proc. of the 18th International Conference on Data Engineering. California, USA: 2002.

[3] Boldi P, Codenotti B, Santini M, et al. Crawler[J]. Software: Practice and Experience, 2004, 34(8): 711-722.

[4] 张　岭, 叶允明, 宋　晖, 等. 一种高性能分布式 Web Crawler 的设计与实现[J]. 上海交通大学学报, 2004, 38(1): 59-63.

[5] 叶允明, 于　水, 马范援. 分布式 Crawler 的研究：结构、算法和策略[J]. 电子学报, 2002, 30(12A): 45-50.