第 34 卷 第 21 期 计 算 机 工 程 2008 年 11 月
Vol.34 No.21 Computer Engineering November 2008
·网络与通信· 文章编号：1000—3428(2008)21—0084—02 文献标识码：A 中图分类号：TP393

APM 架构下动态缓存的 BCLFL 策略

施家庆[1]，牛纪桢[2]，王 凡[2]
(1. 大连理工大学软件学院，大连 116621；2. 大连理工大学电子与信息工程学院，大连 116023)

摘 要：对基于 APM 的 Web 服务器架构进行分析，利用动态缓存来提高服务器的响应时间。对于影响服务器动态缓存的 Web 文件大小、文件连接地址和文件修改时间等因素进行测试，根据测试数据进行分析、研究、探索，给出一种基于链接的小文件日志分块缓存(BCLFL)优化策略，并采用双进程进行实际测试，结果表明该策略在很大程度上提高了服务器的性能。
关键词：APM 架构；动态缓存；基于链接的小文件日志分块缓存

BCLFL Strategy of Dynamic Cache in APM Architecture

SHI Jia-qing[1], NIU Ji-zhen[2], WANG Fan[2]
(1. School of Software, Dalian University of Technology, Dalian 116621;
2. Department of Electronics and Information Engineering, Dalian University of Technology, Dalian 116023)

【Abstract】 By doing some analyses of the APM based Web server, this paper uses the dynamic cache to reduce the response time of the server. Some experiments are done on the factors which are related to the performance of the server with dynamic cache. These factors are the Web file size, the link of the Web file and the modification time of the Web file. Based on the analyzing results of the experiments, a BCLFL strategy is given to solve the problems that are found. A new method which adopts two-process architecture to test the Web system performance is used to test the BCLFL strategy system. The results show that the system is totally improved.
【Key words】 APM architecture; dynamic cache; BCLFL

当前Web已经成为一种主流的发布信息方式，浏览器也成为使用最广泛的客户端，如何进一步提高Web服务器性能及高可用性已成为当前热门的研究课题之一[1]。通过将Web文件缓存在浏览器或者服务器端的Cache技术是解决Web服务响应时间的一种有效途径[2]，本文给出了Apache, PHP, MySQL(APM)架构下一种基于链接的小文件日志分块缓存(BCLFL)优化策略。

1 影响 Web 服务器响应时间的因素

1.1 实验对象

Web 服务的主要任务就是发布信息，因此研究中采用了 Web 服务中最常见的 CMS(Content Management System)。Content 数是 CMS 系统在实验中的重要参数，代表 CMS 中一类文章的数目。而根据大多数网站的特点，CMS 系统中文章的分类在本文中默认为 4 类，这样如果 Content 数为 100，则 CMS 系统中文章的总数就是 400。文中以下的实验将直接针对 Content 数目进行设置。

1.2 文件大小与响应时间的关系

在服务器解析文件的过程中，文件的大小与响应时间关系如下：

$$Cor(s,d)=\frac{Cov(s,d)}{\sqrt{Var(s)\cdot Var(d)}} \quad (1)$$

其中，$Cov(s,d)$是文件大小和响应时间的协方差；$Var(s)$是文件大小的变化；$Var(d)$是响应时间的变化。$Cor(s,d)$表示从服务器获得文档的大小与响应时间之间的关系。当 $Cor(s,d)=1$ 时表示文件大小与响应时间为线性相关。而 $Cor(s,d)=0$ 时表示文件大小与响应时间没有关系，就是说在 $Cor(s,d)=0$ 时无需考虑文件的大小与响应时间的关系。根据文献[3]中的实验结果，$Cor(s,d)$大约在 0.214 5，也就是说缓存文件的大小与响应时间之间的关系不大。

下面通过一个实例验证这个结论。其中 Content 取值分别为：1, 10, 100, 1 000, 10 000，利用 1 000 次 Request 和 50 个 Concurrent，测试结果如图 1 所示。

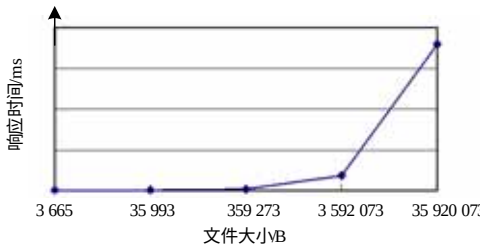

图 1 文件大小与响应时间

由图 1 可知，在文件尺寸大于约 3 MB 时响应时间将快速上涨。中国互联网络信息中心的第 19 次中国互联网报告即《中国互联网络发展状况统计报告(2007/1)》中显示，全国网页数为 44.7 亿个；总计网页字节数约为 122 TB；平均每个网站的网页字节数约为 137 MB；平均每个网页字节数 27.3 KB。这种情况下，网站的响应时间与文件大小的关系不是很大。但是需要控制网站中某个页面不要出现数据量过大的情况，以免影响网站的整体性能。

1.3 Header 设置

Apache 服务器采用一种称为 TTL-based 的持续算法。

作者简介：施家庆(1983－)，男，硕士研究生，主研方向：LAMP软件架构；牛纪桢，副教授；王 凡，讲师、博士研究生
收稿日期：2007-12-27 **E-mail：**shijiaqing@163.com

TTL 用 Expire-now 来设置网页的生命周期。如果在被访问页面文件中没有设置生命周期参数,则 TTL 算法用 fudge_factor *(now - Last-Modify)来设置 Last-Modified 头信息，其中 fudge_factor 是系统定义的参数。

在 PHP 中可以通过函数 header 来设置被访问页面的生命周期参数。下面利用试验来看一下 header 函数在被访问的服务器文件中的作用。首先利用 header 函数使服务器端和客户端的缓存都禁止使用，取 Content 数为 100，进行 1 000 次 Request 和 50 个 Concurrent 的测试实验。代码实现如下：

```
<?php
header('Expires: Mon, 26 Jan 2007 05:00:00 GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Cache-Control: post-check=0, pre-check=0', FALSE);
header('Pragma: no-cache');
?>
```

接着，从被访问的文件中去掉以上代码，从而使用服务器默认的缓存机制，同样取 Content 数为 100，进行 1 000 次 Request 和 50 个 Concurrent 的测试实验，观察使用 header 与不使用 header 的结果如表 1 所示。

表 1　Header 的使用比较　　ms

类型	100,(1 000,50) Respond Time	1 000,(1 000,50) Respond Time
使用 Apache 默认 Cache 机制	3 406	24 906
禁止使用 Cache	3 953	28 625

由表 1 看出，在 100 条 Content，以 1 000 次 Request 和 50 个 Concurrent 测试时，禁止服务器和客户端使用 Cache 情况下的响应时间要比 Apache 默认使用 Cache 时慢 547 ms，速度降低约 16.05%。而在 Content 增加到 1 000 条(共 4 000 条)时速度慢了 3 719 ms，速度降低了约 14.93%。由此可见，禁止 Cache 的使用会造成服务器的响应时间明显下降，有效地利用 Cache 机制提高响应时间(降低响应时间的值)是一种十分可行的途径。

1.4　文件更新频率

为使 Cache 机制更有效地发挥作用，可以考虑先设置 Cache 中一个比较重要的参数 Expire Data，即文件的更新频率。由于对于不同的信息系统及应用领域，被访问文件的更新频率没有一个固定的值，因此参数设置较难。

综上，本文设计了一个基于链接的小文件日志分块缓存(Block Cache which is based on Link, small File and Log, BCLFL)优化策略来有效解决上述问题。

2　BCLFL 策略的设计与实现

BCLFL 策略就是基于链接的小文件日志分块缓存。其中链接是容易被忽视的一个环节，在现代的网页中大量采用了 MVC 模式进行设计和开发。一个文件往往可以处理多个链接，所以需要将一个文件的缓存根据其调用的链接分别进行处理，BCLFL 策略中服务器根据链接来分析获取所需要调用的 Cache 文件。

为了避免 1.2 节中提出的单个文件过大问题，可采用分页机制有效控制文件的大小。事先应充分考虑可能出现的各 Web 页文件大小，从而合理地设定每页 Content 的数量。采用分页机制，需要考虑分页过程中的参数传递方式，其中最常用的方式是利用$_GET 来控制分页参数。BCLFL 策略中使用$_GET 方式来实现分页，$_GET 会使网页的链接改变，这样就转变为链接的问题了。因此，文件过大问题的解决像链接一样只需考虑文件打开时所要调用的 Cache 文件是哪个文件就可以了。

Log 的引入是针对网站的网站管理员而言的，一个动态的网站需要对网站的内容进行更新，这就需要网站管理员对于后台数据库中的数据进行新增、修改和删除。在某个页面被修改后，这个页面的链接所对应 Cache 文件也就应该做相应的更新，不能再采用原先的内容进行更新。本文的 BCLFL 策略中，网站管理员对于数据库的操作也就是网站数据的修改,在数据库中采用一个 Log 表对相关页面和时间进行记录。网站管理员的操作就是前文 1.4 节中所述的文件更新频率，可在数据库中用时间戳(time stamp)进行记录，以便获取文件更新的参考值，从而决定是否需要更新 Cache 文件还是直接读取 Cache 文件。

绝大多数 PHP 网站都使用了 include()或者 require()方法来包含几个网页文件都需要共同包含的相同文件(例如：header.php, footer.php, left.php, right.php 等)。对于这样的页面就需要进行分块，分别保持这些页面的 Cache 值，而不是与包含它的文件使用同一个 Cache 文件。一个显示的页面事实上可能使用了多个 Cache 文件来组合完成，这样的分块其实在一定的程度上也缩小了单个文件的大小，从而避免了单个文件过大问题。每个服务器 Web 文件被服务器解析的流程如图 2 所示。

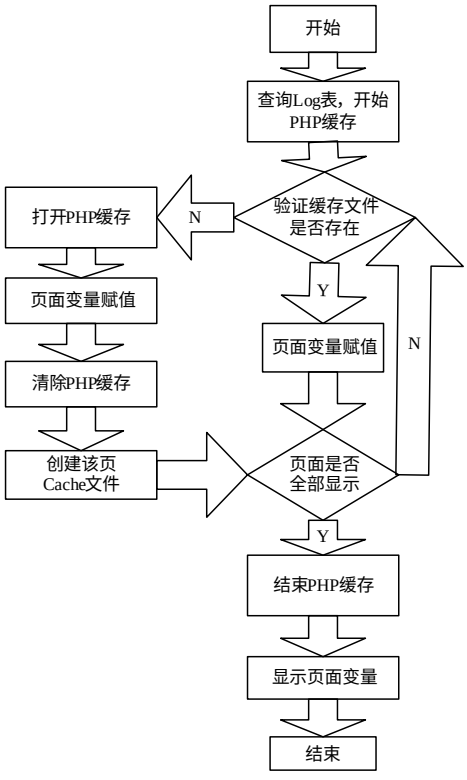

图 2　BCLFL 策略流程

在流程中“验证缓存文件是否存在”是策略中最关键的步骤，该步骤中首先需要根据链接准确地定位 Cache 文件的文件名，再从数据库的 Log 表中得到该页的修改时间，最后验证 Cache 文件修改时间是不是比 Log 中的时间晚，来判断当前链接对应的 Cache 文件是否为数据最后一次修改后所产生的(其中时间的比较是利用 Unix 的时间戳进行的)。如果 Cache 文件不是最新的则重写 Cache 文件，否则就使用该 Cache 文件。

（下转第 88 页）