# adPD:一种速度自适应的动态并行下载技术

周 旭 卢显良 侯孟书 詹 川
(电子科技大学计算机学院 成都610054)

**摘 要** 本文在介绍了现有的并行下载算法的基础上提出了一种新的速度自适应的动态并行下载机制--adPD。adPD通过为速度不同的连接动态分配大小不同的下载任务,可以很好地适应传输连接速度的变化,做到按速度比例分配下载任务量,充分利用带宽。同时,通过划分大小不固定的文件分块,adPD还可以尽可能地减少发送数据请求的数量,缩短请求等待的空闲时间,在减轻提供服务的节点的负载的同时,提高了下载速度。最后,通过实验结果分析了adPD的实际性能,验证了adPD是一种高效的并行下载算法。

**关键词** 并行下载,动态,自适应

## A Speed-Based Adaptive Dynamic Parallel Downloading Technique

ZHOU Xu LU Xian-Liang HOU Meng-Shu ZHAN Chuan
(Department of Computer Science of UEST of China,Chengdu610054)

**Abstract** In this paper, we describe adPD, an improved parallel downloading approach to retrieving files by establishing parallel connections with multiple copies. adPD assigns each server equal portion of file as large as possible, which ensures client downloading from each server without interrupt. And after faster server finish its own work, adPD will reallocate part of unfinished work of some slower server to the faster server. By this means, adPD can dynamically adjust the proportion of a file retrieved from each server during the downloading process, and can also reduce the number of block requests. Experiment results show that adPD is an effective parallel download scheme especially in Peer-to-Peer environments. Our experiment results show that adPD is an effective parallel download scheme.

**Keywords** Parallel download, Dynamic, Adaptive

## 1 引言

为了提高互连网中文件的下载速度,人们做出了种种努力。大型的Internet站点通常为自己建立多个镜像站点[1],用户可以手动或者由软件自动[2]选择性能最好速度最快的镜像,下载所需文件。而在大型的分布式共享存储系统或者Peer-to-Peer系统中[3],一个文件也通常存在多个副本,并分布在不同的机器上,用户可以选择速度快的主机下载自己所需的文件。这些方法都是"一对一"(point to point)模式的单线下载(Single Downloading, SD),服务器的选择对最终下载性能影响很大,而设计一个优化的服务器选择算法比较困难[14]。

为了充分利用多个文件副本和网络带宽,避免复杂的服务器选择过程,最大可能地加快下载速度,A. Miu[5]等人提出了一种新的并行下载技术,其基本思想是客户机同时与多个镜像服务器建立连接,并将文件分为几块分别从不同的服务器下载,当各个部分下载完成时,再由客户机将多个分块合并得到完整的文件。由于同时从多个服务器下载数据,并行下载技术不仅大大加快了文件下载速度,而且避免了复杂的服务器选择过程,增强了下载进程抵抗单线连接失败的能力。由于具有种种优点,并行下载技术在近几年得到了广泛深入的研究,也得到了实际的应用[6~8]。

本文在现有的并行下载算法的基础上,提出了一种改进的速度自适应的动态并行下载机制--adPD。采用adPD算法,客户机可以精确地根据不同服务器的速度情况,动态划分文件块,并选择对应的服务器下载,以达到充分利用带宽、加速下载的目的。adPD还可以动态调整分配给各服务器的分块大小,让速度快的服务器主动分担速度慢的服务器的任务,使得各服务器可以根据自己的能力承担相应比例的下载任务,合理分担了负载,也最大限度地利用了服务器能力。

## 2 相关工作

为了下载一个特定的文件,在确定一组服务器节点后,客户机必须将所需的文件数据分块并分配给相应的服务器来下载。显然,在这个过程中,文件的分块和任务的分配将影响算法实现的复杂度、网络和服务器负载,最终决定文件的下载速度。根据这两个决定因素,可以将现有的并行下载算法分为以下几类:

· Static-Equal:在这种方式中,客户端根据文件大小将文件平均划分为$n$块,分别从$n$个服务器下载,等全部分块下载完毕之后,将各分块合并得到完整的文件数据。"Equal"表示文件被平均划分,"Static"表示各个文件分块的下载任务在开始传输之前就固定地分配给了相应的服务器,并且在传输过程中,这种分配不再改变。

· Static-Unequal:在这种方式中,客户端在开始下载之前,先预测各个服务器的速度,根据预测,将文件按比例划分成大小不等的分块并分配给相应的服务器执行下载任务,等待所有分块下载完成之后,拼合数据得到文件。

· Dynamic:与前面两种静态分割文件的并行下载方法有所不同,Dynamic并不试图在下载之前就决定为每个服务器分配的任务比例,而是将文件分为大小相等的若干小分片。客户端每次向服务器请求不同的分片,当某个服务器的当前分

片下载完成之后，客户端再向该服务器请求另一个分片。当所有分片下载完成之后，客户机拼合分片得到文件。

在上面的三种并行下载方法中，Static-Equal 实现起来最为简单[10]，但也有着明显的缺点：对一个特定的客户端来说，网络上不同服务器的速度之间可能相差几倍到几十倍[9]，这样，Static-Equal 完成下载的时间就完全取决于速度最慢的那个服务器。实验表明，相比在多个服务器中随机选择一个进行单线下载的 SD 方式，在很多时候，Static-Equal 的性能仅仅只有少许提高，甚至有些时候下载时间还长于 SD 方式[12]。

Static-Unequal 克服了前一种方法在划分文件分块上简单平分的缺点，性能比 Static-Equal 有了不少提高。在这种方法中最关键的是对服务器速度的预测，预测的准确与否直接影响到任务的分配以及最后下载时间的长短。由于处在 Internet 中的服务器受到多种因素的影响，性能和速度可能时刻处于变化之中[11]，要想在下载之初精确地预测服务器的速度十分困难。实验表明，由于服务器速度的不确定性，有些时候较大的文件分块反而被分配给了速度较慢的服务器，造成了性能的降低[10]。

与前面两种并行下载方式相比，Dynamic 的最大优势在于：从每个服务器下载的数据量的比例可以较好地与各服务器在下载期间速度的比例相吻合，适应下载过程中带宽和服务器负载的变化。实验表明，Dynamic 方法拥有三种方法之中最好的下载性能[11]。Dynamic 方法中最关键的是要确定一个合适的分片长度 $l$，一个小的 $l$ 值可以使得文件可以以更细的精度在各服务器之间分配，使得各服务器下载数据量的比例尽可能地靠近服务器之间速度的比例。但是过小的 $l$ 值也会造成客户端频繁的向服务器发出数据请求，给服务器和网络带来额外的负担。

## 3 改进的并行下载算法 adPD

### 3.1 并行下载的模型基础及分析

前面介绍的几种并行下载算法在一定程度上较大地提高了客户机下载文件的速度[4, 5]。但是，由于大量的资源集中在少数服务器中，如果所有用户都使用并行下载方式的话，服务器端将建立大量的连接并要同时处理大量的数据请求，引起负荷急剧增加。实验表明，在服务器的负载较低的时候，并行下载的性能较好，但随着服务器负载的增加，并行下载的用户数量越多，客户机体验到的下载性能反而呈下降趋势[10]。为了进一步提高并行下载的性能，并尽可能减少服务器的负载，我们将建立数学模型来分析哪些因素决定了并行下载的性能。

设网络中某客户机需要读取一个文件 $f$，该文件大小为 $F$，并在系统中存在 $m$ 个副本，分别存放在 $m$ 个服务器 $(server_1,\cdots,server_m)$ 上。在并行下载模式下，该客户机同时打开 $m$ 条连接，同时从 $m$ 个服务器下载数据，设客户机到 $server_i$ 的下载速度为 $v_i, i=1,\cdots,m$。设到下载结束时，客户机从各个服务器处下载的数据分别为 $F_i, i=1,\cdots,m$，则可以求得节点从 $server_i$ 处下载完 $F_i$ 需要的时间 $t_i$ 为：

$$t_i=F_i/v_i+t_i^{connect}+n_i t_i^{idle} \tag{1}$$

式(1)中，$t_i^{connect}$ 为客户机与 $server_i$ 建立连接所需的时间。$t_i^{idle}$ 为每次数据请求的响应时间，即客户机向 $server_i$ 发送数据请求到开始接收到数据之间的时间，在这段时间中，数据连接处于空闲状态，没有得到利用。$n_i$ 为客户机从 $server_i$ 处下载 $F_i$ 发送的数据请求次数。在实际网络中，$t_i^{idal}$ 等于客户机到 $server_i$ 的 RTT(round trip time)，这个值和 $t_i^{connect}$ 一样，无法人为控制。所以，为了尽量缩短 $t_i$，减少连接空闲时间，我们必须减少客户机向 $server_i$ 发送数据请求的次数 $n_i$，尽量不打断一个连续的下载过程。

知道了每条连接下载完成的时间，可以求出客户机从 $m$ 个服务器并行下载完文件 $f$ 所需的时间 $T$：

$$T=\max\{t_i\}\forall\, i\in(1,\cdots,m)=\max\{F_i/v_i+t_i^{connect}+n_i t_i^{idal}\},\quad \forall\, i\in(1,\cdots,m) \tag{2}$$

从式(2)可知，当 $t_1=t_2=\cdots=t_m$ 时，$T$ 达到最小值，即如果所有连接同时结束下载，此时的下载时间最短。所以，动态并行下载算法优化下载时间的核心在于合理分配从各服务器下载的数据量，使下载数据量的比例接近或等于从各服务器下载平均速度的比例。如果 Dynamic 算法中分片长度为 $l$，则一共需要发出请求 $F/l$ 次，$l$ 越小，次数越多。对于下载速度越快的连接来说，客户机通过该连接发出请求的次数越多，频率越高，对应的 $n_i t_i^{idle}$ 也就越大，使得该链接下载的平均速度受到影响。另一方面，对下载速度慢的连接来说，由于它的 RTT 时间也相应比较长[13]，$t_i^{idle}$ 值较大，如果请求次数稍多，将明显降低下载的平均速度。从以上分析知道，$n_i t_i^{idle}$ 是动态并行下载算法为了适应连接速度变化而额外付出的开销，要想提高动态并行下载的性能，必须尽量减少这个开销。

为了减少数据请求带来的时间开销，一种方法是动态地调整分块的大小，对速度快的连接使用大的分块，对速度慢的连接使用小的分块[5, 11]。但是由于动态增加分块大小虽然可以减少发出请求的次数 $n_i$，但是同时也增加了文件划分的粒度，很难达到所有连接同时完成下载的目的。另一种方法是使用 pipeline(流水线)的方法[12]，客户机在当前的数据块下载还没有完成的时候，提前发出对下一数据块的请求，新的请求等待(pending)在服务器端，上一数据块的传输一结束，马上就可以进行新的数据块的传输。这种方法可以有效地减少请求之间等待的时间，但是性能的提高依赖于对 RTT 值的精确估计。在实际的网络环境中，服务器负载以及网络情况随时处在变化之中，使得 RTT 值的预计十分困难。并且大量频繁的文件数据请求也会进一步增加服务器负担，从而抵消掉一部分 pipeline 带来的性能提升。

所以，我们认为要提高并行下载的性能，同时不给节点和网络带来过高的负担，最关键的还是减少下载过程中发出的数据请求的次数。

### 3.2 adPD 算法描述

为了让多条连接同时完成数据的下载，必须要尽可能细致地划分分配给各条连接的下载数据量。要达到这个目的，动态并行下载算法必须使用更小的文件分片，这又将带来数据请求数的增加。为了解决这个矛盾，我们提出了一种速度自适应的动态并行下载算法 adPD。adPD 并不预先确定固定的文件分片，而是先将文件平分，为每条连接分配相等大小的数据块并行下载，当速度快的连接先下载完自己的那部分数据后，再从比它慢的连接那里分担一部分数据的下载任务，通过快慢协作，达到减少数据请求次数，提高下载速度的效果。下面我们详细描述 adPD 的算法流程。

设文件 $f$ 大小为 $F$，存放在系统中 $m$ 个服务器上，设某客户机到每个服务器的下载速度为 $v_i(t), i=1,\cdots,m$，速度随时间变化而变化。客户机按照如下流程，同时打开 $m$ 条连接从各服务器并行下载数据：

1)客户机根据文件大小 $F$，将文件平均分为 $m$ 份，分别从各服务器并行下载。

2)在时刻 $t$，当某条连接 $i$ 完成自己的任务后，客户机将其他连接未完成的部分按照一定的比例划分给该连接，继续下载。分配步骤如下：

a)计算各连接的加权平均速度 $v_j=\alpha F_j/t+(1-\alpha)v_j^{new}$，$0<\alpha<1$，$j=1,\cdots,m$。其中 $F_j$ 是到目前为止，连接 $j$ 下载的总数据量，$F_j/t$ 为连接 $j$ 的平均下载速度，$V_j^{new}=v_j(t)$ 是连接 $j$ 当前的下载速度。$\alpha$ 决定了平均速度在 $v_j$ 中所占的权重。

b)设连接 $j$ 尚未完成的下载任务为 $\overline{F}_j$。记速度比连接 $i$ 慢的连接集合为 $M$，即 $M=\{j|v_j<v_i,\overline{F}_j>0,j\neq i\}$。若 $M\neq\varnothing$，计算 $M$ 中各条连接完成目前剩下的下载任务量 $\overline{F}_j$ 所需的时间：$t_j=\overline{F}_j/v_j$，$j\in M$。

c)选择 $M$ 中剩下时间最长的连接 $j$：$j\in M$，使得对 $\forall k\in M,k\neq j$，有 $t_j>t_k$。将连接 $j$ 剩下的任务量 $\overline{F}_j$ 划分为两块 $\overline{F}_i$、$\overline{F}_j$，重新分配给连接 $i$、$j$，分配公式如下：

$$\overline{F}_i/\overline{F}_j=v_i/v_j,\overline{F}_j=\overline{F}_j+\overline{F}_i \quad (3)$$

d)若 $M=\varnothing$，设 $M'$ 为所有速度比连接 $i$ 快的连接的集合：$M'=\{j|v_j>v_i,\overline{F}_j>0,j\neq i\}$。连接 $i$ 选择 $M'$ 中速度最慢的一条连接 $j$，按照式(3)重新为连接 $i$、$j$ 分配下载任务。

3)为了避免文件分块过细，设置一个阈值 $B$。如果一个连接 $j$ 剩下的任务量 $\overline{F}_j\leqslant B$，则

a)如果有速度比自己慢的连接要求分担自己的任务，连接 $j$ 拒绝这个要求。

b)如果有速度比自己快的连接要求分担自己的任务，连接 $j$ 放弃自己剩下的下载任务，交给比自己快的连接下载。

4)如果各连接下载完毕，客户机重组获得的文件分片，得到完整文件。

### 3.3 adPD 并行算法优点

adPD 与其他并行下载算法相比，有很多优点。首先，与一般的动态并行下载算法固定分块大小不同，adPD 实现了分块大小的动态调整：在下载之初采用大的分块，随着下载的进行，分块逐渐变小，最大可能地解决了下载任务分配精度和数据请求次数之间的矛盾。如果把一般的动态并行下载算法称为“Dynamic-Equal”(Equal 指分块的大小固定)，那么 adPD 可以称为“Dynamic-Unequal”。

adPD 的动态分块方法也保证了每条连接尽可能地连续下载，不被打断。对于速度慢的连接来说，如果采用 Dynamic 并行下载算法，每次下完一个固定的小分块再发送新的请求，实际上是在做“加法”，而采用 adPD 并行下载算法，预先分配的一块大的下载任务不断被速度快的连接分担，任务量逐渐减少，是在做“减法”。在这种情况下，速度慢的连接通常不需要发送新的数据请求，只需要连续地下载自己的任务直到结束，这可以有效地避免慢连接的数据请求延迟带来的下载平均速度降低。这对于速度快的连接来说，可以在不受影响的情况下连续下载完分给自己的第一块数据，然后在每一次请求中都尽可能多地分担速度慢的连接的任务，下载的连续性得到了最大程度的满足，从而大大减少了发送数据请求的次数。

adPD 也可以很好地适应各连接间速度的差异。对于连接之间速度相差不大的情况，第一次均分就有可能满足任务的分配，不需要更多的数据请求。对于连接之间速度相差大的情况，adPD 也能根据连接之间的速度的比例，精确地划分下载任务，以充分利用带宽。如果一条连接在下载期间速度变化比较大，adPD 可以通过连接之间互相分担任务来动态地调节任务分配比例。在这种最坏的情况下，adPD 发送的数据请求数量会增加，但不会超过 Dynamic 方法。

另外，adPD 通过3.2小节中第3)步的操作，可以使得文件的最后一块数据始终由速度快的连接来下载，避免了可能出现的由慢连接下载最后的数据块引起的速度减慢的弊端[5]，这种情况通常被称作是“idle time in downloading the last block”。

## 4 实验结果

### 4.1 实验环境及参数

为了验证 adPD 算法的有效性和性能，我们用 java 实现了 adPD 动态并行下载算法，并对“驱动之家”网站[1]的多个镜像服务器进行了并行下载试验。为了测试 adPD 在不同文件大小下性能的差异，我们总共对三种尺寸的文件作了实验，文件大小分别是600k、2M 和10M。为了测试并行程度 dop (degree of parallel)对下载的影响，我们对 dop 从2到8都进行了实验。另外，文件分块的阈值 $B$ 设为一般 Dynamic 算法采用的分块大小50k[11]，公式中的速度预测权重 $\alpha$ 设为0.59[11]。

### 4.2 实验结果

首先我们测试并行下载性能随 dop 的变化情况。图1分别是下载文件大小为10M、2M、600k 时 adPD、Dynamic 算法和 Static-Unequal 算法(S-U PD)在不同 dop 值下平均下载时间的比较图。从图中可以看出，adPD 在三种情况下，下载时间都比其他两种短。在 dop 大于5之后，adPD 的性能提高不明显，一个原因是因为客户端带宽有限，在下载速度接近这个极限时，继续增加并行度，下载速度也很难再继续提高。而当文件较小(2M 和 600k)时，Dynamic 和 Static-Unequal 的性能接近，Static-Unequal 甚至比 Dynamic 的性能还略好(图1.c)，这主要是因为文件较小，同样50k 的文件分片相比之下对任务的调节能力比不上在文件大的时候那么明显，在这种情况下，简单的下载方法反而更加有效。

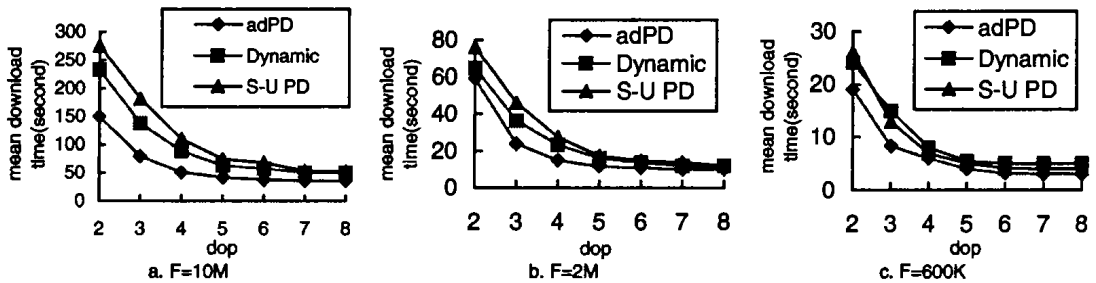


图1 不同文件大小情况下三种并行下载算法性能比较

图2是在不同文件大小的情况下，adPD 发送的平均数据请求数量随 dop 值的变化。我们可以看到，随着下载并行度的增加，主机发送的消息数呈增加趋势。这主要是因为下载连接越多，每次分给连接的任务数据量就越小，连接完成自己任务的速度加快，导致发送请求的频率和数量增加，在一定程度上抵消掉了 adPD 带来的性能提高。这也是当并行度增加到5以后，adPD 性能提高不明显的另一个原因。而对于 Dynamic 算法来说，由于它的文件分块固定为50k，故发送的数据请求数量是固定的，分别为12、41和205次，远远高于 adPD，特别是在

(下转第177页)

请 BeanLock2。如果申请成功，那么必然存在一个 Tx1-BeanLock1-Tx2-BeanLock2-Tx1的环，出现死锁。此时唯一的方法是事务超时回滚，造成资源的浪费。但是并发控制机制能够察觉到死锁的出现，因此 Tx2对 BeanLock2的申请不会成功，而会导致它的回滚。

使用加锁机制，虽然牺牲了持久化管理器的效率，增加了复杂度，但是保证了实体 Bean 的可靠性，确保复杂的网络环境中实体 Bean 也能正确地工作。

**小结** 本文介绍了实体 Bean 容器对 CMP 的支持。为了支持 CMP，实体 Bean 容器需要提供持久化管理器。持久化管理器实现实体 Bean 的静态映射和动态映射。本文描述的持久化管理器使用较好的映射模型，并且具有较高的效率。此外，实体 Bean 容器提供并发控制机制管理对实体 Bean 的并发访问。本文描写的实体 Bean 容器和 CMP 持久化管理器已在中科院软件所自主开发的 J2EE 应用服务器[7]中实现。下一步的工作包括扩展 OR 映射模型，以体现更多的 OO 编程的思想；扩展对数据库的访问接口，以更好地支持数据库的新特性。

## 参 考 文 献


1 Sun Microsystems. Java 2 Platform Enterprise Edition Specification, v1.3. (2001/3/30)
2 Sun Microsystems. Enterprise JavaBeans Specification, Version 2.1. (2003/6/3)
3 Buschmann F, Meunier R, Rohnert H, et al. A Systems of Patterns: Pattern-Oriented Software Architecture. New York: John Wiley & Sons Ltd, 1996
4 Keller W. Mapping Objects to Tables-A pattern language, 2003, 7
5 Ambler S W. Building Object Applications That Work-Your Step-by-Step Handbook for Developing Robust Systems With Object Technology. New York: SIGS Books/Cambridge University Press, 1998
6 Ambler S W. Process Patterns: Building Large-Scale Systems Using Object Technology. New York: SIGS Books/Cambridge University Press, 1998
7 范国闯. Web 应用服务器关键技术研究


(上接第170页)

大文件(10M)的时候，Dynamic 的请求次数是 adPD 的5.1倍。大量的数据请求不仅给服务器带来了较大的负载，也使得下载性能大大下降。

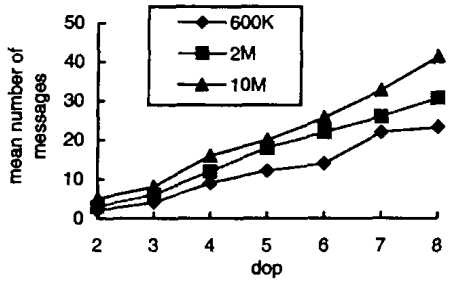


图2 不同文件大小情况下 adPD 平均数据请求数量随 dop 值变化图

在3.1节中我们提到了采用 pipeline 的方法避免数据请求带来的连接空闲。为了验证 pipeline 带来的性能提高，我们对 Dynamic 和 adPD 在文件大小为10M 的情况下使用了 pipeline 方法，试验结果见图3。从图中可以看到，使用 Pipeline 之后，Dynamic 性能提高明显，平均等待时间下降了不少，但是性能仍然没有超过没有使用 pipeline 的 adPD 算法。这主要是因为网络节点间的 RTT 值变动比较大，这给精确估计发送请求的提前量带来了困难，使得一部分请求的延迟没有完全被 pipeline 消化。而使用 pipeline 并不能给 adPD 带来多少性能提高，这主要是因为 adPD 的数据请求数量已经相对较少，pipeline 能够节省下来的空闲时间不是很多。考虑到使用 pipeline 给算法增加了额外的复杂度，我们认为在实际使用中一般的 adPD 算法就已经足够了。

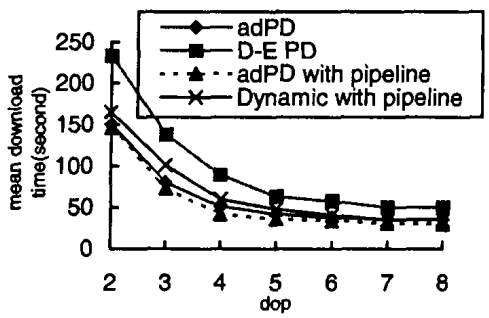


图3 使用 pipeline 前后 adPD 与 Dynamic 性能对比图 (F=10M)

**结论** 本文提出了一种新的速度自适应的动态并行下载机制--adPD。adPD 通过为速度不同的连接动态分配大小不同的下载任务，可以很好地适应传输连接速度的变化，做到按速度比例分配下载任务量，充分利用带宽。同时，通过划分大小不固定的文件分块，adPD 还可以尽可能地减少发送数据请求的数量，在减轻提供服务的节点的负载的同时，提高了下载速度。实验证明 adPD 与已有的几种并行下载算法相比，性能提高明显，缩短了下载时间，同时也大大减少了发送的数据请求的数量，是一种高性能的并行下载算法。

## 参 考 文 献


1 Mydrivers' web site. http://www.mydrivers.com
2 Akamai's web site. http://www.akamai.com
3 Gnutella. http://gnutella.wego.com
4 Rodriguez P, Kirpal A, Biersack E W. Parallel-access for mirror sites in the internet. In: Proc. of IEEE INFOCOM 2000, March 2000
5 Miu A, Shih E. Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet. url: http://nms.lcs.mit.edu/?aklmiu/comet/paraload.html, December 1999
6 Speedbit's download accelerator. http://www.speedbit.com
7 Cohen B. Incentives Build Robustness in BitTorrent. http://bitconjurer.org/BitTorrent/documentation.html, May 2003
8 OpenCola Swarmcast. http://www.opencola.org/projects/swarmcast.shtml
9 Myers A, Dinda P A, Zhang H. Performance characteristics of mirror servers on the internet. In INFOCOM (1), 1999. 304～312
10 Gkantsidis C, Ammar M, Zegura E. On the Effect of Large-Scale Deployment of Parallel Downloading. In: IEEE Workshop on Internet Applications (WIAPP'03), 2003
11 Rodriguez P, Biersack E. Dynamic parallelaccess to replicated content in the Internet. IEEE/ACM Trans. on networking, 2002, 10(4)
12 Koo S G M, Rosenberg C, Xu D. Analysis of Parallel Downloading for Large File Distribution. In: Proc. of FTDCS 2003, San Juan, Puerto Rico, May 2003
13 Zeitoun A, Jamjoom H, El-Gendy M. Scalable Parallel-Access For Mirrored Servers. In: The 20th IASTED Intl. Conf. on Applied Informatics (AI 2002), Innsbruck, Austria, Feb. 2002
14 Sayal M, Breibart Y, Scheuermann P, Vingralek R. Selection Algorithm for Replicated Web Servers. In: Workshop on Internet Server Performance, SIGMETRICS, Madison, USA, June 1998