AmazingStore 离线下载系统中的资源存储策略

（北京大学网络与信息系统研究所，北京 100871）

摘要：离线下载系统是基于云技术的分布式系统，用于提供代理下载服务。系统中，完成的任务资源被保存在存储区，便于其他用户提交相同任务时进行资源复用。离线下载系统的这一特性提高了资源的可用性，节约了用户的时间。随着离线下载系统的普及，用户不断增多，需要存储的资源飞速增长，对系统的存储空间提出挑战。如何在有限的存储空间中，淘汰老旧资源，存储更有价值的资源，成为本文探讨的存储策略问题。本文就 AmazingStore 离线下载系统提出多种存储策略，并使用基于真实用户下载日志整理而成的测试数据进行性能评测。通过评测，分析了各种存储策略的优劣，找到了理想的策略。

关键词：存储策略；离线下载；云计算；分布式系统；P2P

中图分类号：TP393

Resource Storage Strategy on AmazingStore Off-line Download System

CAO Yue, LI Zhenhua, DAI Yafei

(Institute of Network Computing and Information Systems, Peking University, Beijing 100871)

Abstract: Off-line Download System is a distributed system based on Cloud Technology, providing downloading service for clients. User submits tasks to system and waits for dragging files back until tasks are accomplished by the system. Files required by clients are stored in storage pool after downloading in order to be reused when required again by other clients, thus saves users' downloading time as well as energy. However, the capacity of storage pool is limited and cannot meet the aggressive increase of task volume required by clients. A sophisticated storage strategy is called for to help the system stores the most valuable files.In this paper, after brief introduction, several strategies are evaluated by data set obtained from user download log, and the most ideal strategy is found after deep analysis on the performance evaluation.

Key words: Storage Strategy; Off-line download; Cloud Computing; Distributed System; P2P

0 引言

随着网络的普及，对个人用户而言，计算机服务已经渐渐转变为网络服务。仅仅需要一台性能适中、存储空间不大的普通电脑，甚至是笔记本电脑，用户便能借助网络体验高性能、拥有大规模存储空间的服务器所提供的服务。

另一方面，伴随着计算机硬件技术的发展，服务器成本降低而单台服务器性能提升有限，越来越多的网络公司开始采用基于大规模计算机集群的云技术。云技术将用户计算机的任务集中在云服务器中批量完成，为用户提供服务的同时，也因为其自身的规模效应提高了整体效率，减少了资源与能源的浪费。

2009 年，迅雷、腾讯公司先后开始提供基于云技术的离线下载服务[1] [2]，又称云下载。离线下载使用系统服务器为用户提供了一种较以往更加便捷、高效、经济的下载方式，得到用户的喜爱。随着离线下载的普及，用户数目日益增多，下载数据高速增长，对离线下载系

作者简介：曹越（1986-），男，北京大学硕士，主要研究方向：分布式系统

通信联系人：代亚非(1958-)，女，教授，主要研究方向：网络信息与分布式系统. E-mail: dyf@pku.edu.cn

统提出了挑战。

离线下载技术的特点在于通过存储已下载完成的资源提高资源的可用性，减少用户的下载等待时间。然而新资源不断推出，存储空间有限，系统必须进行选择性的存储与淘汰。在有限的存储空间下，找到一种尽可能多的提高资源复用次数的方法，成为我们的研究动机。

本文以资源的命中率、加权命中率为目标，提出各种资源存储策略，模拟评测后进行深入分析，得到详细的结论。

1 资源存储策略与评测

1.1 系统介绍

1.1.1 P2P 网络与 AmazingStore

P2P（Peer to Peer）网络又称对等网络，是一种分布式网络。网络中，节点以对等方式相连，每个节点既是资源的提供者，又是资源的接受者。这种非中心化的特点为 P2P 网络带来了高度的可扩展性和健壮性。

AmazingStore[3]是一个基于 P2P 与云技术的分布式系统，用于资源共享与存储。由北京大学网络实验室开发，主要使用在教育网环境，用户大多为高校学生与老师。共享的资源大多为学习资源、视频音乐文件、游戏程序与常用软件。平均 15 天内活跃用户数 3 千人，日下载次数 5 千次，日下载文件总量 1TB。

1.1.2 AmazingStore 离线下载系统

使用离线下载系统，用户仅仅需要将任务提交给云服务器，由服务器替用户完成下载任务，无需在线等待。任务提交过程可以在较差的网络环境下完成，例如断断续续的无线 wifi 或者低速的临时网络，为用户提供了方便。下载完成后，用户再从服务器将文件取回。取回过程的文件传输速度仅仅取决于用户的本地带宽。高效经济的下载方式同时节约了用户的时间与资源。

本文中提及的离线下载系统基于已有的 P2P 网络：AmazingStore，由 Web 服务器，下载服务器（Task Worker）以及存储区（Storage）组成。

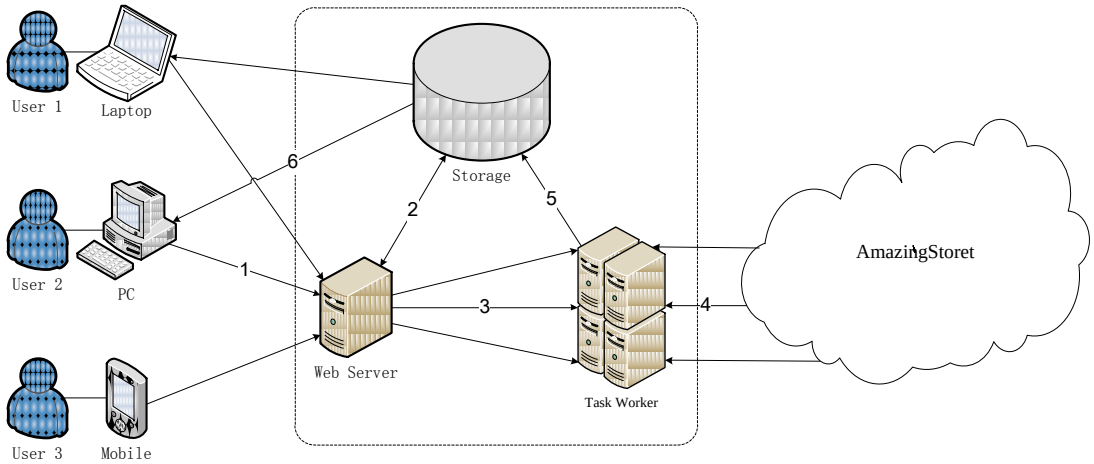

图 1 离线下载系统结构图

1.2 策略目标

离线下载系统对任务资源进行存储，目的在于提高资源的复用率。较高的资源复用率

有两大优点：

- 节约用户下载时间，增加用户满意程度。
- 节约系统资源。通过省去重复下载，节约了系统的带宽与负载。

与以上两个优点相对应，我们提出两个衡量存储策略优劣的目标：命中率与加权命中率。我们定义命中率为用户提交的所有任务中，被命中次数所占的百分比：

$$命中率 = \frac{命中次数}{总任务数} \times 100\%$$

系统维持较高的命中率，对用户而言则有较大的概率无需等待系统执行下载任务，从而节约用户时间，提升用户的满意程度，达到离线下载系统设计的初衷。简要的说，命中率是依据命中的次数来计算的。

针对命中率忽略了文件大小的缺陷，考虑为每次命中按文件大小增加权重，我们定义加权命中率：

$$加权命中率 = \sum_{所有命中任务} \left(\frac{文件大小}{总任务大小} \times 1 \right) \times 100\%$$

或者将上式整理为：

$$加权命中率 = \frac{命中文件大小}{总任务大小} \times 100\%$$

对于没有命中的任务文件，系统需要将其作为新文件存储。因此，我们定义新文件率：

$$新文件率 = \frac{新文件大小}{总任务大小} \times 100\%$$

后文的分析中，通过对每日新文件大小、新文件率的计算，间接计算出了加权命中率。

加权命中率客观的反应了系统的负载和带宽因为对任务文件进行存储而得到节约的程度，也真实的反应了用户下载等待时间的节约程度。简要的说，加权命中率是依据命中的流量来计算的。

1.3 测试数据介绍

为了评测存储策略，本文使用相同数据对不同策略进行模拟实验，再将结果进行对比。数据取自 AmazingStore 的用户下载日志：从 2010 年 1 月 23 日至 2011 年 2 月 20 日，共 394 天。数据包含 209 万次下载任务，其中 12 万个不同文件，平均每日 5343 次下载任务。任务总大小 406.8TB，去掉重复文件后总文件大小 30.3TB，平均每日 1.04TB 任务。394 天中，平均单个任务大小 204MB。

其中，低于平均大小 204MB 的文件个数占总文件个数 76.8%，低于平均大小的任务个数占总任务个数 73.5%。测试数据中小文件远多于大文件。

文件平均访问次数 16.89 次。其中，访问次数大于等于 13 次的文件，占所有文件 20.11%，而命中了访问次数大于等于 13 次的文件的任务，占总任务 87.65%。命中率分布符合二八定律。

如果系统不采用任何选择策略而存储所有任务文件，41.07%的文件被存储后再也没有被访问过。这些文件的存储对离线下载系统没有意义，不加以选择的存储浪费了存储空间。文件个数占总任务数 5.92%，意味着 5.92%的任务为系统添加了新文件，其余任务均为重复访问已存储文件，系统理想命中率为 94.08%。以上数据说明，采用合理的策略对

任务进行选择性存储，可以减少绝大多数重复下载工作。

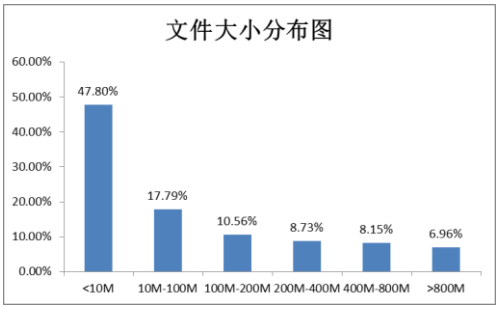

图 2 文件大小分布图

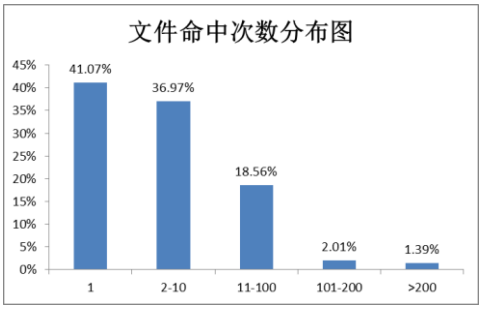

图 3 文件命中次数分布图

1.4 基于队列操作实现的存储策略

三大传统内存替换策略 FIFO、LRU、LFU[4]与张晓东提出的 LIRS 策略[5]都可以作为离线下载系统的存储策略。以上四种策略的特点在于使用队列（或者栈）来实现。

将上述四种策略应用于我们的测试数据，设置存储区大小从 0.2TB 至 10TB，分别测试结果如图 4。

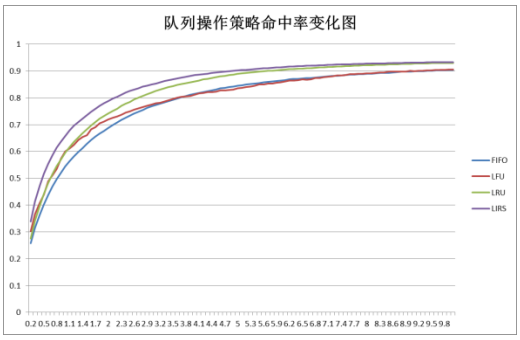

图 4 队列操作策略命中率图

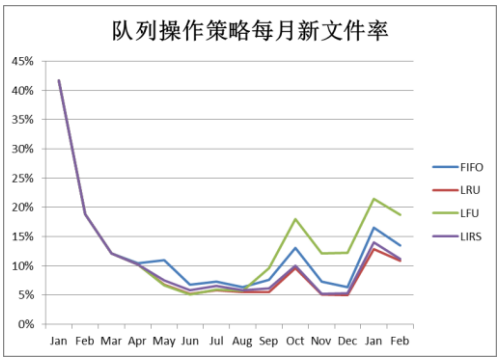

图 5 队列操作策略每月新文件率图

当存储空间较大时，LFU 每次淘汰命中次数为 0（访问次数为 1）的资源。这些资源在队列中按 FIFO 排列，导致 LFU 的表现与 FIFO 接近。另一方面，过去的流行资源长期占据存储空间不被淘汰，导致命中次数为 0 的队列能用的存储空间远小于 FIFO 策略中使用的空间。因此 LFU 的表现甚至差于 FIFO。以上可以归纳为访问次数带来的存储毒化问题（Storage Pollution）。

图 5 显示 2010 年 9 月新任务的大量出现时，LFU 受到较大冲击，FIFO 次之，LRU 与 LIRS 相当。此处，LFU 的存储毒化问题再次显露。LFU 关注资源的访问次数，假设过去的流行资源在未来也会有更高访问频率。由于这个假设在我们的测试环境中不再有效，因此 LFU 在面对大量新任务出现时，表现不如其他策略。

由于在存储区大小超过 5TB 以后，各个策略的命中率提升有限，以下设定存储区大小为 5TB 进行分析。

表 1 队列操作存储策略命中率与加权命中率表

序号	代号	命中率	加权命中率
1	FIFO	84.43%	84.11%
2	LRU	88.92%	87.60%
3	LFU	83.62%	78.87%
4	LIRS	90.25%	87.52%

四种基于队列操作实现的存储策略命中率与加权命中率如表 1 所示，从中得到与前文的分析一致的结论：

1. LIRS 的命中率显著高于其他三种，加权命中率略低于 LRU。

2. LRU 命中率与加权命中率均高于 LFU，说明偏重于最近访问时间（Recency）的策略比偏重于累计访问次数（Frequency）的策略在我们的用户数据中具有更好的表现。

1.5 基于价值函数的策略

除了通过操作队列与栈来实现存储策略，也可以使用价值函数来实现策略：计算不同资源的价值，并将价值排序。通过不断维护队列，保持任何时刻队尾资源为最无价值资源。使用不同的价值函数，能够实现不同的存储策略。

淘汰策略可以划分为基于最近访问时间的策略、基于累计访问次数的策略以及同时基于上述两点的策略[6]。在对存储空间中的资源进行价值计算时，考虑资源的新旧程度、流行程度和存储资源的代价，分别对应以下三个参数：

表 2 价值函数参数表

参数	描述
r	文件最近一次访问（命中）距离现在的时间
f	文件累计访问次数
s	文件大小

其中，参数 r 的值为最近访问日期与当天间隔的天数加 1。参数 f 的值为命中次数加 1，对于刚被存储的资源，初值为 1。参数 s 的值为文件的兆字节（MB）数。

1.5.1 考虑文件大小

当文件大小一致，或集中于平均值附近呈正态分布而趋于一致时，考虑文件大小对结果的影响不大。然而在我们的用户日志中小于 10MB 的文件占 47.8%，任务占 30.6%。小于平均大小的文件占 76.8%，任务占 73.5%。因此，我们的离线下载系统工作环境并不满足正态分布条件，需要将文件大小纳入考虑。

我们并非简单的去掉同等情况下体积最大的文件[7]，而是将文件大小作为价值函数的参数，对传统存储策略改进得到三种新的策略：

表 3 价值函数策略与队列操作策略命中率与加权命中率表

基于价值函数的策略					基于队列操作的策略			
	代号	价值函数	命中率	加权命中率		代号	命中率	加权命中率
5	RN	$V(r,s)=\frac{1}{r\times s}$	93.01%	85.78%	2	LRU	88.92%	87.60%
6	FN0	$V(f,s)=\frac{f}{s}$	92.48%	82.64%	3	LFU	83.62%	78.87%
7	BN0	$V(r,f,s)=\frac{f}{r\times s}$	93.17%	87.02%				

从测试结果中可以得到两个结论：

1. 考虑文件大小能够显著提高策略命中率。

2. 综合考虑最近访问时间与访问次数，能够显著提高策略命中率与加权命中率。

由于价值函数的使用，除了能够引入文件大小（s）作为参数，原本难以兼顾的两个参数：访问次数（f）与最近访问时间（r）也可以有效结合在一起考虑。这是基于队列操作的策略所不具备的优势。

1.5.2 分组淘汰

将文件大小作为参数计算资源的价值，会导致大文件价值过低。例如，大小为 2GB 的安装文件，除非在访问次数与最近访问时间两项参数拥有极大的优势，按价值函数计算出的价值会远远低于 2MB 大小的文档。

一种合理的解决方案，是将资源分组，例如 PSS 策略[8]，LRU-SP 策略[9]与 CSS 策略[10]。根据价值在组内进行排序，每一组内的资源进行排序，不同组之间的资源互不影响。

在我们的离线下载系统中，用户下载的文件按大小可以分为以下四类：

表 4 用户下载资源分组比例表

组	大小	个数	次数	体积	流量	命中率	加权命中率
1	<10M	47.80%	30.63%	0.31%	0.24%	90.76%	90.28%
2	10M-100M	17.79%	26.28%	2.97%	6.08%	95.99%	96.37%
3	100M-1G	28.78%	40.02%	39.02%	62.81%	95.74%	95.39%
4	>1G	5.63%	3.07%	57.70%	30.87%	89.15%	86.12%

通过分组，每组内资源大小（s）差别不超过 10 倍，与访问次数（f）和最近访问时间（r）两个参数相结合计算得到的价值更为合理。分组后，将存储区分为四个区域分别存储对应的资源。

四个区域的划分比例，可以参考测试数据中四种资源的比例。由于划分空间涉及任务或文件的大小，因此根据体积或流量划分存储区空间更为合理。此外，不同组间各自的命中率（或加权命中率）也不相同，可以按根据加权命中率加权后的流量进行划分。分别对不同划分方式进行测试后发现，按流量划分比按体积划分拥有更高的命中率与加权命中率，按加权命中率对流量进行加权后再据此比例划分存储区能进一步提高命中率。

将按文件大小分组的策略运用与于上一小节中提出的三种策略，得到 RY、FY0、BY0 三种策略。对三种策略进行测试，得到命中率与加权命中率如下表：

表 5 分组策略与不分组策略命中率与加权命中率表

分组策略				不分组策略			
	代码	命中率	加权命中率		代码	命中率	加权命中率
8	RY	90.62%	86.88%	5	RN	93.01%	85.78%
9	FY0	89.55%	84.05%	6	FN0	92.48%	82.64%
10	BY0	91.21%	87.89%	7	BN0	93.17%	87.02%

上一小节里，通过引入文件大小作为决策因子（价值函数的参数），存储策略有效的提升了命中率，却损失了加权命中率。本节提出按文件大小分组，在提升了系统命中率的同时，也增加了加权命中率。分组策略是对考虑文件大小策略的有效补充。

1.5.3 老化机制

曾经的热门文件因为具有较高的访问次数而始终获得较高的价值评分，从而占据存储区无法被淘汰的问题，我们称作存储区毒化问题（Storage Pollution）。解决存储区毒化问题的方法，是引入合理的老化机制（Aging Mechanism）。受α -Aging 策略[11]与 LFU-Aging 策略[12]启发，我们提出两种策略：

- 每隔固定时间（比如 1 天），将所有资源的访问次数减少一定比例：

$$f_{i+1} = \alpha \times f_i$$

其中老化因子α 满足 0 < α < 1（例如，在后文的策略中尝试取α = 0.965）。

- 设定一个阈值，当所有资源的平均访问次数达到该阈值时，所有资源的访问次数

减少一定比例：

$$f_{i+1} = \alpha \times f_i$$

其中老化因子α 满足 0 < α < 1（在后文的策略中，尝试取α = 0.5）。

以上两种策略中，除了访问次数定时减少，每次访问时访问次数还是加 1。然而按比例减少会导致文件访问次数（f）不再是整数，因此也将它称为文件流行度评分，可以视作广义的文件访问次数。

分别对前文所述的四种涉及访问次数的策略（FN0、FY0、BN0、BY0）使用两种老化机制，得到策略 FN1、FN2、FY1、FY2、BN1、BN2、BY1、BY2。

对于第一种策略，如何选取合适的老化因子α 成为影响策略命中率与加权命中率的关键。我们通过分析访问次数最多的文件来确定老化因子：访问次数最多的文件在 394 天内达到 2386 次访问。在前 197 天时，累计访问 1104 次，假设从此不再被访问，如果存在一个老化因子能够使其在此后 197 天访问次数逐渐降至新文件水平：访问次数为 1，则我们称这样的老化因子为理想老化因子。

即理想老化因子应满足等式：

$$\text{半期访问次数} \times \alpha^{\text{半期时间}} = 1$$

该文件半期访问次数为 1104 次，半期时间为 197 天，计算得到α =0.9651。

对于第二种策略，和第一种的差别在于老化速度取决于整体访问次数。例如，寒暑假里用户减少，每日任务数量减少，则老化速度也减慢。

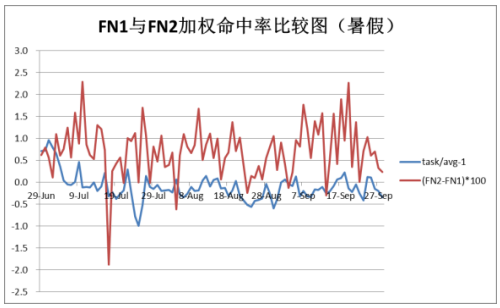

图 6 两种老化策略比较图（暑假）

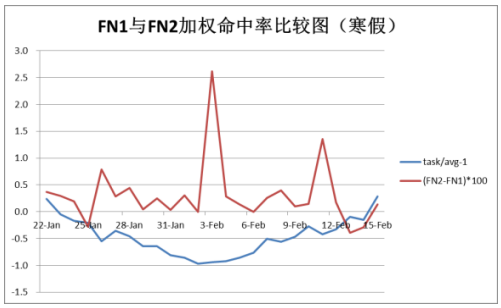

图 7 两种老化策略比较图（寒假）

以上两图中，用 FN2 的加权命中率减去 FN1 的加权命中率，再放大 100 倍作为对比数据（即图中红线），蓝线为当日任务数与平均任务数之比减 1。可以看出，暑假里大多数时间 FN2 的加权命中率较 FN1 高出 1 至 2 个百分点，寒假里 FN2 的加权命中率也优于 FN1。结据此我们得出论，第二种老化策略能够明显提高寒暑假期间的加权命中率。

对本节所提出八种策略进行测试，命中率与加权命中率如下表所示：

表 6 两种老化机制命中率与加权命中率表

	老化机制 1				老化机制 2			
		代码	命中率	加权命中率		代码	命中率	加权命中率
不分组	11	FN1	93.01%	84.67%	15	FN2	93.02%	85.02%
	12	BN1	93.02%	87.70%	16	BN2	92.89%	87.77%
分组	13	FY1	90.84%	86.29%	17	FY2	90.93%	87.34%
	14	BY1	90.97%	88.00%	18	BY2	90.82%	87.93%

从表中可以得到以下结论：

1. 对于仅考虑访问次数的策略，老化机制 1 比无老化机制在命中率和加权命中率两

方面都更优。老化机制 2 比老化机制 1 两方面都更优。

2. 对于同时考虑访问次数和最近访问时间的策略，在命中率方面老化机制 2 不如老化机制 1，老化机制 1 不如无老化机制。但在加权命中率方面，两种老化机制都优于无老化机制。

1.6 双存储区策略

受 LIRS 策略双队列设计方式的启发，我们提出一种双存储区策略：设立临时存储区用于存储最近加入的文件，另设长期存储区用于存储从临时存储区使用基于价值函数的策略挑选而出的文件。

临时存储区大小占整体的 20%，其中的文件采用 FIFO 方式定期淘汰。文件存放在临时存储区的时间称为考察期，在我们的系统中设为 7 天。

长期存储区采用基于价值函数的策略进行淘汰，其中价值函数为：

$$V(r, f, s) = \frac{f}{r \times s}$$

其中，根据访问次数（f）采用老化机制情况分为三种子策略。根据是否采用分组淘汰机制再分为两类。采用双存储区模式的策略共有六种，对其进行模拟测试得到命中率与加权命中率如下表：

表 7 六种双存储策略命中率与加权命中率表

老化		不分组				分组		
		代码	命中率	加权命中率		代码	命中率	加权命中率
0	19	DN0	93.42%	88.76%	22	DY0	92.05%	89.32%
1	20	DN1	93.27%	89.08%	23	DY1	91.86%	89.30%
2	21	DN2	93.17%	89.09%	24	DY2	91.69%	89.20%

从上表中对比可以对出结论：在使用相同的价值函数与老化机制时，无论是否采用分组机制，双存储区策略使命中率与加权命中率都能得到提升。

2 结论

本文首先提出了存储策略的目标：命中率与加权命中率。其中命中率按命中次数计算，加权命中率按命中流量计算。较高的命中率给用户带来较好的体验，而较高的加权命中率能较多的节约系统资源，减少用户下载等待时间。

在简要介绍测试数据后，引入了四种基于队列操作实现的存储策略，并提出三种考虑文件大小、基于价值函数的存储策略。通过对以上策略进行测试分析，发现考虑文件大小确实能提高命中率。而价值函数的使用，让同时考虑文件访问次数和最近访问时间成为了可能。测试发现，同时考虑两个因素的确能够有效提高命中率和加权命中率。

由于引入了文件大小作为价值函数的参数，导致命中率提高、加权命中率降低。针对这一问题，本文随后提出了按文件大小分组淘汰的方法。通过测试，发现按任务流量对存储区分组能够达到更高的命中率和加权命中率。

由于文件访问次数引起的存储区毒化问题，本文随后提出通过加入两种老化机制来解决。对引入不同老化机制的策略进行测试，结果验证了老化机制的意义。

最后，本文提出双存储区策略，并根据是否引入老化机制、引入何种老化机制派生出六种子策略。进行测试后，找到了命中率与加权命中率单项指标最高的策略与综合最优的

策略：具有最高命中率的策略是 DN0：93.42%，具有最高加权命中率的策略是 DY0：89.32%，综合考虑命中率和加权命中率，最优的策略为 DN1，两项指标分别为 93.27%与 89.08%。

[参考文献] (References)

[1] 迅雷离线下载网址:http://vip.xunlei.com/freedom/lixian.html?cachetime=1304508239273
[2] 腾讯旋风离线下载网址：http://xf.qq.com/help_video.html
[3] Zhi Yang, Ben Y. Zhao, Yuanjian Xing, Song Ding, Feng Xiao and Yafei Dai，AmazingStore: Available, Low-cost Online Storage Service Using Cloudlets, In Proceedings of The 9th International Workshop on Peer-to-Peer Systems (IPTPS2010) San Jose, CA, April 2010
[4] Podlipnig and L. Böszörmenyi, A survey of web cache replacement strategies, ACM Comp. Surveys 35 (4) (2003), pp. 374-398
[5] Song Jiang , Xiaodong Zhang, LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance, Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, June 15-19, 2002, Marina Del Rey, California
[6] JIN, BESTAVROS. GreedyDual: Web caching algorithms exploiting the two sources of temporal locality in Web request streams. In Proceedings of the 5th International Web Caching and Content Delivery Workshop.
[7] WILLIAMS, ABRAMS, STANDRIDGE, ABDULLA, FOX. Removal policies in network caches for World-WideWeb documents.In Proceedings of ACM SIGCOMM. ACM Press, New York, New York, NY, 293-305.
[8] AGGARWAL, WOLF, and YU. Caching on the World Wide Web. IEEE Trans.Knowl. Data Eng. 11, 1 (Jan.), 94-107.
[9] CHENG, KAMBAYASHI. A sizeadjusted and popularity-aware LRU replacement algorithm for Web caching. In Proceedings of the 24th International Computer Software and Applications Conference (COMPSAC).
[10] TATARINOV. An efficient LFU-like policy for Web caches. Tech. Rep. NDSU-CSORTR-98-01, Computer Science Department, North Dakota State University, Wahpeton, ND.
[11] ZHANG, J., IZMAILOV, R., REININGER, D., AND OTT, M. 1999. Web caching framework: Analytical models and beyound. In Proceedings of the IEEE Workshop on Internet Applications.
[12] ARLITT, M. F., CHERKASOVA, L., DILLEY, J., FRIEDRICH, R. J., AND JIN, T. Y. 2000. Evaluating content management techniques for Web proxy caches. ACM SIGMETRICS Perform. Eval. Rev. 27, 4 (Mar.), 3–11.