

面向存储服务的分布式缓存系统研究

王 侃, 陈志奎

(大连理工大学软件学院, 大连 116620)

摘 要: 针对分布式环境下高频率异地数据访问造成的系统性能下降的问题, 对 SBM 模型进行改进, 提出分布式缓存系统 D-Cache, 给出基于最优价值度的缓存文件替换算法——OCV。数字模拟实验结果证明, 与 DartCache 系统相比, D-Cache 系统能更有效地减少系统访问延迟, 增加吞吐量, 提高分布式环境下系统的性能。

关键词: 分布式缓存系统; 缓存管理; 替换算法

Research on Storage Service-oriented Distributed Cache System

WANG Kan, CHEN Zhi-kui

(Software College, Dalian University of Technology, Dalian 116620)

【Abstract】 In the distributed environment, the high frequency of remote data access leads to a decline in system performance. Aiming at this problem, this paper improves the SBM model, and designs a Distributed Cache system(D-Cache). A new cache replacement algorithm, OCV is given. Digital simulation results demonstrate that D-Cache system can shorten the response time of system and increase the throughput of system more effectively, which improves performance in the distributed environment, compared with the distributed cache system DartCache.

【Key words】 distributed cache system; cache management; replacement algorithm

1 概述

随着现代企业应用规模的扩大, 越来越多的企业采用了分布式的管理模式。在分布式的企业管理模式中, 企业总部与成员企业通过高速互联网进行连接。每个成员企业都是能够完成特定业务的单位, 拥有专用的 Web 服务器、应用服务器、元数据管理服务器和 SAN 存储网络。成员企业构架如图 1 所示。在分布式的存储环境下, 用户需要频繁地进行异地访问。频繁的数据访问会带来信息传输上延迟的增加和性能下降, 使用合适的缓存技术能够有效地解决该问题^[1]。

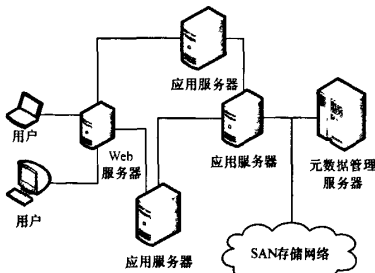


图1 成员企业构架

2 相关技术

2.1 SBM 模型的改进

面向存储服务的缓存管理模型 SBM^[2]能够针对不同的应用对缓存进行有效的管理。但是在分布式的存储环境下, SBM 存在如下缺陷: 在分布式环境下, 每一类存储应用都对应多个缓存对象, 而管理对象针对单一缓存对象的管理, 并不能将所有相关类型的缓存充分利用, 没有达到最高的资源利用率; 缓存分配与应用无关, 所有的缓存对象都通过唯一的缓存分配对象竞争缓存资源, 势必会造成与应用的脱离, 不能

达到最佳的缓存分配策略。

针对上述缺陷, 本文提出以下 2 个改进方法:

(1) 在缓存管理模块增加全局缓存文件管理功能, 用于存储管理本地缓存文件的信息。

(2) 构建虚拟缓存池, 统一管理缓存空间。

2.2 D-Cache 系统架构

将改进后的 SBM 模型应用到分布式缓存系统 D-Cache 中, 得到 D-Cache 的架构, 如图 2 所示。

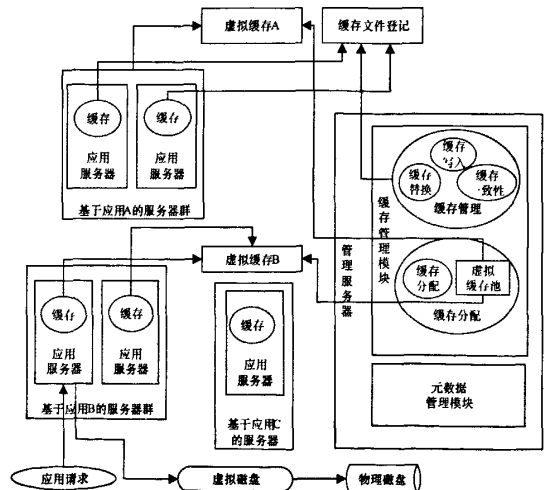


图2 D-Cache 系统架构

作者简介: 王 侃(1987—), 男, 硕士研究生, 主研方向: 网络存储, XML 数据库; 陈志奎, 教授、博士

收稿日期: 2010-03-15 E-mail: zkchen@dlut.edu.cn

缓存区位于各应用服务器，直接与应用关联。管理模块位于管理服务器，由缓存控制和缓存分配 2 个子模块组成。

2.2.1 缓存控制模块

缓存控制模块为缓存分区提供基于某种应用的缓存管理功能。在缓存资源给定的前提下，缓存控制按功能划分为 3 个部分：

- (1)缓存写入：从存储介质中读取相应的数据块并置于应用服务器的缓存中。
- (2)一致性操作：将缓存中修改过的数据块同步写入存储介质中，控制缓存与存储介质中的数据一致性。
- (3)缓存回收：当缓存空间不足时，从缓存中替换出相应缓存块。

缓存控制模块根据应用的具体需求，为相关应用服务器提供所需的缓存管理功能，同一缓存控制服务器可以同时服务于多个应用服务器缓存。

2.2.2 缓存分配模块

缓存分配模块针对每一种应用，将空闲缓存资源以虚拟缓存池的形式统一管理。类似于存储的虚拟化，虚拟缓存池由每一类应用的缓存区虚拟化形成。

应用服务器在读写分区缓存空间过程中按需求从共享缓存池中申请新的缓存资源，新分配的缓存块由缓存控制模块通过合适的替换算法加入到相应的缓存空间中。

当某一种应用的虚拟缓存池中的空闲缓存资源不足时，缓存分配模块发出回收操作，由控制模块控制缓存资源的回收。缓存回收算法和缓存替换算法分别决定了应从的缓存空间中回收哪些缓存资源和从缓存空间中替换出哪些缓存块。

3 分布式缓存的数据描述

针对 SBM 模型的改进，本文设计了 2 个表：虚拟缓存表和缓存文件登记表。前者用于存储空闲缓存的元数据信息；后者用于存放所有缓存文件的元数据信息。2 个表分别存储在管理服务器的缓存分配模块和缓存控制模块。

3.1 虚拟缓存表

管理服务器的缓存分配模块能够对本地所有的空闲缓存区进行统一的管理。对缓存区信息的描述，则通过虚拟缓存表来实现。根据虚拟缓存表，将基于同一种应用的空间统一起来，建立虚拟缓存池，管理空闲的缓存空间。虚拟缓存表的格式如表 1 所示，其中，可用度属性反映了缓存区可利用的程度，由特定的缓存区选择算法确定。

表 1 虚拟缓存表格式

表项名称	表项含义
缓存编号	缓存的唯一标识
应用服务器编号	缓存所在应用服务器的编号
缓存大小	缓存空间总大小
剩余空间大小	缓存实际可用的空间大小
地址	缓存所在的物理地址
所属应用类型	所在应用服务器的应用类型
Availability	缓存可被利用的程度

3.2 缓存文件登记表

管理服务器的缓存管理模块保存所有缓存文件的记录，称之为缓存文件登记表。通过登记表，可以查找本地以及全局范围内的缓存文件。在不考虑缓存 I/O 速率因素的前提下，本文设计了缓存文件登记表，格式如表 2 所示，其中，缓存文件名是由缓存文件地址通过 MD5 算法^[3]得到的散列值。采用这种方法，有效解决了缓存文件多副本的问题，保证了文

件名的唯一性。另外，本文加入能够表示文件重要性的 value 值，根据 value 的取值采用合适的替换策略。

表 2 缓存文件登记表格式

表项名称	表项含义
缓存文件名	文件地址的 MD5 散列值，唯一标识文件
文件类型	文件的类型，如文本、图片等
文件大小	文件的大小，物理块的整数倍
文件地址	文件对应的首块地址
访问次数	文件已经被访问过的次数
最近访问时间	该文件最后一次被访问的时间
生命时间	缓存文件的有效时间
所在 cache 号及地址	存储该文件的缓存编号
Cache 大小	存储该文件的缓存大小
所属应用类型	所在应用服务器的应用类型
Value 值	该缓存文件的价值度

当处理用户请求的应用服务器没有在缓存中搜索到所需文件，应用服务器将请求转发到管理服务器的缓存管理模块。缓存管理模块首先搜索登记在缓存文件登记表中的文件，如果有，则将文件请求转发给拥有该文件的应用服务器，由该应用服务器完成用户请求的处理，将所需文件返还给用户。

4 缓存管理调度算法

4.1 缓存区的选择算法

根据 D-Cache 系统的设计，系统每次缓存文件时，需要通过虚拟缓存表的可用度值，从所有空闲缓存区进行选择合适的缓存以存储文件。下面给出可用度值的计算方法。

影响缓存区选择的因素有：缓存空间的大小(cacheSize)，剩余容量的大小(restSize)，应用服务的类型，最近被访问的时间(lastAccessTime)，以及被访问的次数(accessCount)、成本和年龄等。

对于某一类应用，假设相关应用服务器有 n 台，给出可用度计算公式：

$$Availability(i) = \frac{restSize(i) \times accessCount(i) \times cacheSize(i)}{lastAccessTime(i)}$$
$$i=1,2,\dots,n$$

其中，Availability(i) 为缓存 i 的可用度；cacheSize(i) 为服务器缓存 i 的总大小；restSize(i) 为应用服务器的缓存剩余空间大小；accessCount(i) 为访问的次数；lastAccessTime(i) 为最近被访问的时间，accessCount(i)/lastAccessTime(i) 在一定程度上反映出缓存访问的频率。

将所有应用服务器按照可用度值进行排序，取优先级最高的一个服务器缓存进行存储。倘若所有的服务器缓存都已经空间不足，则将调用最适合该应用服务器的替换算法进行替换。

4.2 缓存文件替换算法

4.2.1 传统的替换算法

缓存的性能在很大程度上依赖于缓存的替换算法。传统的替换算法，例如 LRU, LFU, SIZE，仅仅是考虑单一的影响因素，不能达到最合适的替换，而 GDS, GDSF 虽然考虑了多方面的影响因素，但由于算法本身的原因，仍不能够达到理想的替换。

文献[4]设计了基于最小延迟代价的替换策略 LLC(Least Latency Cost)算法。该算法针对字节延迟提出，增强了 Web 环境中的公平性。但 LLC 仍存在着不足之处：由于算法是以访问对象出现的频率为权，这样根据算法，极易将访问频繁的对象替换出缓存，不符合实际情况。另外，在缓存系统中，并不是每次访问结束，都将访问的文件存入缓存，需要根据

实际情况决定是否替换现有文件。

基于这个观点,本文根据 LLC 算法,提出基于最优缓存价值度的缓存替换算法 OCV(Optimal Cache Value)。

4.2.2 缓存文件替换算法 OCV

定义 1 延迟代价(cost): 延迟代价指数据的获取代价。当用户请求的对象在缓存中,称为缓存命中,设取该对象的延迟代价为 v_i ; 当用户请求的对象不在本地缓存,需要向存储系统获取,设此时获取第 i 个对象的延迟时间为 b_i , 则有:

$$cost_i = \begin{cases} v_i, i \in S \\ b_i, i \notin S \end{cases}$$

定义 2 价值度(value): 本文用价值度来形容对象的重要性,表示为

$$value_i = \frac{f_i \times Rf_i}{cost_i \times size_i}, Rf_i = \frac{Tu_i - Tm_i}{Tn_i - Tu_i}$$

其中, f_i 为对象的访问频率; Rf 为缓存对象的剩余寿命; Tu_i 为缓存数据的上次更新时间; Tm_i 为缓存数据的上次修改时间; Tn_i 为当前时间; $size_i$ 为对象的大小。价值度越高,说明该文件越重要,越不应该从缓存中替换出来。

定义 3 系统价值度(Value): 所有缓存对象的带权价值之和,表示为

$$Value = \sum_{i=1}^S F_i \times value_i x$$

其中, F_i 为权值,通常为 1。如果系统价值度越高,则说明系统的缓存策略越成功。本文设计的替换算法的目标,就是尽量使系统的价值度达到最高。令 $F_i=1$, 根据定义 2 和定义 3, 可以得到:

$$Value = \sum_{i \in S} \frac{f_i \times Rf_i}{v_i \times size_i}$$

当系统缓存区发生变化时, $Value$ 相应改变取值。假设某时刻系统 $Value_k$ 是最优, V_{k+1} 是被替换出缓存的对象, 替换后腾出的空间存放 d , 则下一时刻:

$$Value_{k+1} = \sum_{i \in S} value_i \neq value_k + value_d - \sum_{i \in V_{k+1}} value_i = Value_k + \frac{f_d \times Rf_d}{v_i \times size_d} - \sum_{i \in V_{k+1}} \frac{f_i \times Rf_i}{v_i \times size_i}$$

由于替换算法无法控制 $Value_k + value_d$ 的数值, 因此当 $\sum_{i \in V_{k+1}} value_i$ 最小值时, 则系统的 $Value$ 得到最大值。系统首先判断 $Value_{k+1}$ 与 $Value_k$ 的大小, 决定是否将对象替换, 令 $T = value_d - \sum_{i \in V_{k+1}} value_i$, 得到:

$$Value_{k+1} = \begin{cases} Value_k + T & T > 0 \\ Value_k & T \leq 0 \end{cases}$$

根据缓存文件登记表和虚拟缓存表中的 $Value$ 值和 $Availability$ 的值, 采用合适的策略进行缓存的替换。D-Cache 缓存替换策略算法如下:

Step1 轮流对请求的文档对象进行处理, 令当前请求对象是 d ;
Step2 if IsInCache() //如果 d 已经在缓存

更新缓存中 d 所对应的访问次数, 生命时间以及 $Value$ 值, 并同步更新到

管理服务器;

else ()

{ 查找管理服务器缓存文件登记表;

if IsInLocalCache() //如果 d 在本地其他的缓存中

{ 将访问请求转发到相应的应用服务器, 有该服务

器完成用户访问;

更新缓存中 d 所对应的文件登记表, 并同步更新到管理服务器;

```
}
else//不在本地缓存
{
    查找虚拟缓存表, 返回 Availability 最大的缓存区号;
    while (缓存没有足够空间保存对象)
        for (缓存中所有对象 i)
            { d = MIN( $\sum F_i \times value_i$ );
              替换  $d$  所对应的对象; }
}
```

Step3 将 d 装入缓存;

Step4 设置 d 对应于缓存文件登记表的值。

5 系统性能评价

5.1 与 DartCache 系统的性能比较

文献[5]提出并设计了分布式的缓存系统 DartCache。系统通过应用服务器集群的方式, 将 DartCache 分布到各个集群节点上, 并通过负载均衡器协调各节点的缓存负载。在 DartCache 中, 各个节点之间采用 P2P 的传输协议进行交互, 而且只有通过这些交互实现数据的分布、负载均衡。每次数据的存放都要通过各个节点的交互完成。这样, 每次缓存的操作都需要每个节点的加入, 增加了系统的通信负担。在 D-Cache 的设计中, 数据的分布、负载均衡的控制工作交给了缓存管理模块, 由缓存管理模块统一管理和分配资源, 减轻了应用服务器节点的交互负担。

5.2 模拟实验

本文在实验环境里配置了 3 台台式机和 1 台服务器, 用以模拟应用服务器和管理服务器。测试的系统是笔者实验室开发的购物网上购物系统, 缓存替换算法采用改进的缓存文件替换算法 OCV。通过 JMeter 测试软件, 用多线程模拟多用户访问。实验环境如下:

- (1)硬件环境: 1)应用服务器: Intel(R) Pentium(R) Dual E2180 @ 2.00 GHz CPU, 1 GB 内存, 120 GB 硬盘。2)管理服务器: Intel(R) Core(TM) 2 Duo CPU E4500 @ 2.20 GHz CPU, 2 GB 内存, 160 GB 硬盘。
- (2)操作系统平台: Windows XP Professional 操作系统。
- (3)编程环境: JDK1.6.0_02, Apache Tomcat 6.0。
- (4)测试软件: jmeter-2.3.2。
- (5)测试数据: 访问延迟, 吞吐量。

5.3 实验数据分析

通过实验得到平均的访问延迟数据如图 3 所示。

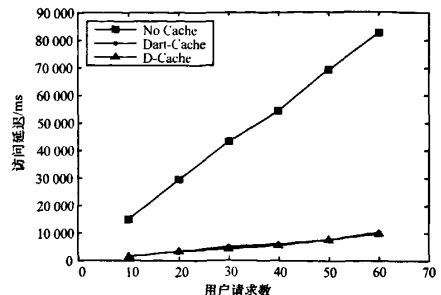


图 3 D-Cache 与 DartCache 系统平均访问延迟的比较

可以看出, 在不使用缓存的情况下, 系统的平均访问延迟 (下转第 85 页)

实验同时将 MLFI 算法与几种多标签算法在 Reuters-21578 数据集上的分类性能作了比较,实验结果如表 4 所示。其中只列出衡量算法综合指标的 F-score, MLFI(N)指不使用校验过程的 MLFI 算法, MLFI(Y)指使用校验过程的 MLFI 算法。

表 4 几种多标签算法的分类性能比较

多标签算法	f-score
J-SVM	0.869 90
LSI-SVM	0.856 90
Binary MFOM	0.881 40
MCMFOM	0.883 23
MLFI(N)	0.851 32
MLFI(Y)	0.886 22

J-SVM, MLFI 算法使用全部特征, LSI-SVM, Binary MFOM 和 MCMFOM 算法使用 LSI 进行特征选择。J-SVM, LSI-SVM, Binary MFOM 算法都使用二元分解方法,不同的是, J-SVM, LSI-SVM 算法使用的 SVM 作为二元分类器, Binary MFOM 使用 MFoM 学习算法。而 MCMFOM 算法提出了一种 MultiClass 分类器来提升 MFoM 在多标签分类中的分类性能。MLFI 算法在校验前 F-score 为 0.851 32, 低于同样使用全部特征的 J-SVM 算法一个百分点,而与使用 LSI 进行特征选择的 LSI-SVM 算法相差不多,但明显低于 Binary MFOM 与 MCMFOM。但是经过校验后 MLFI 算法的 F-score 提升为 0.886 22,比表中分类性能最好的 MCMFOM 高了大约 0.3 个百分点,这从另一角度表明了利用频繁项集对分类结果校验是有效的。

4 结束语

本文提出一种基于频繁项集的多标签文本分类算法。通过对校验前后分类结果的比较,证明了利用类别之间的关联规则来校验分类结果的方法能有效提高分类的查全率,从而达到提高分类器整体性能的目的。与几种多标签分类算法的 F-score 指标进行比较,证明该算法具有较高的分类性能。将挖掘类别频繁项集用于校验分类结果的思想与其他算法结合起来提高分类性能,以及多标签分类过程中阈值的选择等是下一步的研究方向。

参考文献

- [1] Joachims T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features[C]//Proc. of European Conf. on Machine Learning. Chemnitz, Germany: [s. n.], 1998.
- [2] Schapire R E, Singer Y. Boostexter: A Boosting-based System for Text Categorization[J]. Machine Learning, 2000, 39(2/3): 135-168.
- [3] Zhang Mingling, Zhou Zhihua. Multi-label Learning by Instance Differentiation[C]//Proc. of the 22nd AAAI Conference on Artificial Intelligence. Vancouver, Canada: [s. n.], 2007.
- [4] 姜 远, 余俏俏, 黎 铭, 等. 一种直推式多标记文档分类方法[J]. 计算机研究与发展, 2008, 45(11): 1817-1823.
- [5] 眭俊明, 姜 远, 周志华. 基于频繁项集挖掘的贝叶斯分类算法[J]. 计算机研究与发展, 2007, 44(8): 1293-1300.
- [6] Uden M. Rocchio: Relevance Feedback in Learning Classification Algorithms[C]//Proc. of ACM SIGIR Conference on Research and Development in Information Retrieval. Melbourne, Australia: [s. n.], 1998.

编辑 顾皎健

(上接第 82 页)

迟都在 10 000 ms 以上,且随着访问请求数的增多迅速增长,当用户并行访问请求数为 60 时,访问延迟甚至高达 82 475 ms。而使用分布式缓存的 D-Cache 和 DartCache 系统有着明显的优越性,相对不使用缓存的情况平均减少了 89.3%。而 D-Cache 与 Dart-Cache 相比略显优势,比 DartCache 平均减少了 8.87%。

D-Cache 与 DartCache 吞吐量的比较如图 4 所示。

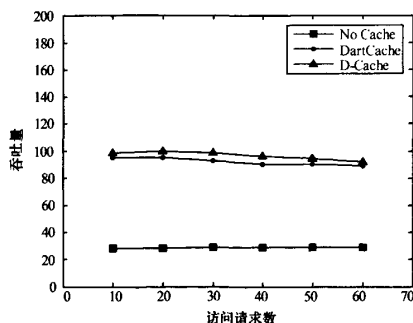


图 4 D-Cache 与 DartCache 吞吐量的比较

由图 4 可以看出,在不使用缓存的情况下,系统的平均吞吐量只有 28.80,且在很小程度上呈现递增的趋势。而使用了分布式缓存的 D-Cache 和 DartCache 系统的平均吞吐量都在 90 以上,分别为 96.74 和 92.04,相对不使用缓存的情况平均提高了 2 倍以上。D-Cache 较之 DartCache 有明显的优势,系统吞吐量平均提高了 5.1%。

由模拟实验结果得知, D-Cache 系统比 D-Cache 系统更有效地提高分布式环境下系统的访问效率。较之 DartCache 系统, D-Cache 平均缩短 8.87% 的响应时间,增大 5.1% 的吞吐量。

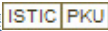
6 结束语

本文通过增加缓存文件和虚拟缓存的管理机制,改进了 SBM 资源利用率不高的缺点。基于最优价值度的思想,提出新的缓存替换算法 OCV,增加了缓存替换的准确性。模拟实验证明,分布式缓存系统 D-Cache 能够有效减轻各服务器的查询负担,提高数据访问的效率,为大型分布式存储管理系统的进一步研究奠定了基础。

参考文献

- [1] Tewari R, Dahlin M, Vin H M, et al. Design Considerations for Distributed Caching on the Internet[C]//Proc. of the 19th IEEE Distributed Computing Systems. Austin, TX, USA: [s. n.], 1999.
- [2] 孟晓恒, 李一鸣, 卜庆忠, 等. 一种面向存储服务的缓存管理模型[J]. 计算机工程, 2009, 35(1): 30-32.
- [3] Rivest R. The MD5 Message-digest Algorithm[R]. RFC 1321, 1992.
- [4] 韩英杰, 石 磊. 基于最小延迟代价的 Web 缓存替换算法研究[J]. 计算机工程与设计, 2008, 29(8): 1925-1928.
- [5] 谢骋超, 陈华钧, 张 宇. DartCache: 一个基于哈希表的分布式 Cache 系统[J]. 计算机科学, 2006, 33(8): 155-161.

编辑 顾皎健

作者: [王侃](#), [陈志奎](#), [WANG Kan](#), [CHEN Zhi-kui](#)
作者单位: [大连理工大学软件学院, 大连, 116620](#)
刊名: [计算机工程](#) 
英文刊名: [COMPUTER ENGINEERING](#)
年, 卷(期): 2010, 36(15)
被引用次数: 1次

参考文献(5条)

1. [Tewari R; Dahlin M; Vin H M](#) [Design Considerations for Distributed Caching on the Internet](#) [外文会议] 1999
2. [孟晓; 李一鸣; 卜庆忠](#) [一种面向存储服务的缓存管理模型](#) [期刊论文] - [计算机工程](#) 2009(01)
3. [Rivest R](#) [RFC 1321. The MD5 Message-digest Algorithm](#) 1992
4. [韩英杰; 石磊](#) [基于最小延迟代价的Web 缓存替换算法研究](#) [期刊论文] - [计算机工程与设计](#) 2008(08)
5. [谢骋超; 陈华钧; 张宇](#) [DartCache: 一个基于哈希表的分布式Cache系统](#) [期刊论文] - [计算机科学](#) 2006(08)

本文读者也读过(6条)

1. [孟阳](#) [分布式缓存系统MCACHE的优化与测试](#) [学位论文] 2010
2. [李文道; 杨小虎](#), [LI Wen-xiao](#), [YANG Xiao-hu](#) [基于分布式缓存的消息中间件存储模型](#) [期刊论文] - [计算机工程](#) 2010, 36(13)
3. [李文中; 陈道蓄; 陆桑璐](#), [LI Wen-Zhong](#), [CHEN Dao-Xu](#), [LU Sang-Lu](#) [分布式缓存系统中一种优化缓存部署的图算法](#) [期刊论文] - [软件学报](#) 2010, 21(7)
4. [苏琳](#) [分布式缓存核心的研究与实现](#) [学位论文] 2010
5. [石晓星; 石磊; 卫琳](#), [SiI Xiao-xing](#), [SHI Lei](#), [WEI Lin](#) [基于背包理论的流媒体缓存算法](#) [期刊论文] - [计算机工程](#) 2010, 36(6)
6. [王小燕](#), [WANG Xiao-yan](#) [一种高效的流媒体代理缓存替换算法](#) [期刊论文] - [计算机工程](#) 2009, 35(14)

引证文献(1条)

1. [李东阳; 刘鹏; 丁科; 田浪军](#) [基于固态硬盘的云存储分布式缓存策略](#) [期刊论文] - [计算机工程](#) 2013(4)

本文链接: http://d.g.wanfangdata.com.cn/Periodical_jsjgc201015028.aspx