Journal of Computer Applications
计算机应用,2013,33(11):3267-3270
ISSN 1001-9081
CODEN JYIIDU
2013-11-01
http://www.joca.cn

文章编号:1001-9081(2013)11-3267-04
doi:10.11772/j.issn.1001-9081.2013.11.3267

移动数据库缓存模型研究

翁唱玲[1],杨　清[2*]
(1.湖南科技大学 计算机科学与工程学院,湖南 湘潭 411201;　2.湖南科技大学 网络中心,湖南 湘潭 411201)
(*通信作者电子邮箱 qyang@hnust.edu.cn)

摘　要:针对移动数据库系统性能有待提高的问题,提出了一种移动数据库缓存模型。采用基于消息摘要的同步算法,通过比较移动客户端与服务器消息摘要表中的消息摘要值,完成缓存同步,维护移动客户端缓存与服务器数据的一致性;该模型还考虑了数据的时效性与事务的优先级,设计了一种基于价值函数的缓存替换算法。实验结果表明,随着缓存数据个数的增加,所提算法的缓存命中率高于最近最少使用(LRU)和LA2U算法,同时随着访问频率的增加,事务的重启率低于LRU和LA2U,有效提高了移动数据库缓存的性能。

关键词:移动数据库;缓存同步;缓存替换;一致性;数据有效性
中图分类号:TP311　　**文献标志码**:A

Research on cache model in mobile database

WENG Changling[1], YANG Qing[2*]
(1. *School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan Hunan* 411201, *China*;
2. *Network Center, Hunan University of Science and Technology, Xiangtan Hunan* 411201, *China*)

Abstract: To improve the performance of mobile database system, a cache model was proposed for mobile database. A kind of synchronization algorithm based on message digest was used in this model. By comparing the value of message digest in mobile client and server, the algorithm completed the cache synchronization, and maintained the consistency of mobile client cache and the data in server. The timeliness of the data and the priority of the transaction were considered in this model. A cache replacement algorithm based on cost function was designed. The experimental results show that the cache hit rate of the proposed algorithm is higher than Least Recently Used (LRU) and Least Access-to-Update Ratio (LA2U) algorithm along with the increase of the number of cache data. At the same time, the restart rate of transaction is lower than LRU and LA2U while the frequency of access increases. The performance of the cache of mobile database is improved.

Key words: mobile database; cache synchronization; cache replacement; coherence; data validity

0　引言

在移动环境中,由于移动设备的移动性以及无线网络的不稳定,会有断连的情况出现,同时,移动设备没有过多的计算能力,并且还依靠电池。由于无线网络较窄的带宽,持久并稳定的访问网络是很难的[1-2]。移动设备在断连时使用缓存数据来处理多种任务。对于移动数据库,支持网络断开操作这一条件是关键[3]。而移动设备的存储能力必定不能与服务器相提并论,因此使用缓存替换技术相当重要。在断连的环境下,移动客户端和服务器都可以对数据进行操作,故两者存在不可避免的冲突。同步技术可以解决数据不一致,并保证数据的完整性,是移动数据库系统中的重要课题[4]。

移动数据库系统中要满足在断连的情况,移动客户端也能够进行操作,需要采用复制缓存技术,与之相关的研究已有一些成果。

文献[5]提出的两级复制考虑了移动数据库计算环境中可信网络部分和无线不可靠网络部分的性能差异,能有效地解决移动计算环境中频繁断连的问题。其缺点是暂态事务要在基节点上重做,增加了系统的负荷。

文献[6]提出了根据数据的读写频率进行复制的动态复制算法。如果在移动数据库系统中对数据的读取很频繁,就选择大量的复制数据副本,这样缓存的命中率大大提高,同时还能减少服务的负载;如果在移动端对数据的更新操作很频繁,此时应该适当降低数据的复制,减少冲突,避免降低数据的更新速度。

文献[7]中的平均机制,通过衡量每个对象的访问频率来决定替换的数据。访问频率也可以通过平均访问时间衡量,当一个对象拥有最高的平均访问时间,则它被替换。这个方法的缺点是它对于数据更新的反应比较迟缓。

Acharya等[8]依据三级复制[9]中所具有的广播环境特点提出了一种实用缓存替换算法——LIX算法。该算法是在LRU(Least Recently Used)的基础上,充分考虑了广播频率对缓存的影响,LIX算法也是采用栈结构,不同于LRU算法的是,每个广播磁盘都有相对应的一个栈,而LIX维护的不是一个栈,是一组栈。缓存对象存放在栈中,栈与广播磁盘是相对应的。如果该缓存对象被命中,则将它移至栈顶。如果缓存没有命中,且缓存已经满了,LIX算法只比较每个栈中栈底对象的LIX值,其中LIX值最小的对象将被替换出缓存。每次缓存替换时,LIX算法的计算量只是一个常数值。当单个移动客户机的访问概率和服务器广播调度中采用的概率分布相差较大时,LIX算法的性能就越明显。

文献[10]中的LA2U(Least Access-to-Update Ratio)缓存

收稿日期:2013-05-31;**修回日期**:2013-08-07。
基金项目:湖南省教育厅重点科学研究项目(10A028);湖南省科技计划项目(2013FJ4050)。
作者简介:翁唱玲(1987-),女,湖南湘潭人,硕士研究生,主要研究方向:移动数据库;　杨清(1969-),男,湖南益阳人,教授,博士,主要研究方向:无线传感器。

替换算法是将客户端对数据的访问次数与服务器对同一数据的更新次数的比值作为衡量标准,选取比值大的数据放入缓存替换比值较小的数据。而 LAUD(Least Access-to-Update Difference)替换策略是将查询频率与修改频率的差值作为衡量标准,选取差值大的数据放入缓存,将差值小的对象替换出去。这两个算法与传统分布式缓存替换的共同点是都考虑了访问频率,不同之处是 LA2U 和 LAUD 还将数据的更新频率列入考虑,因此这两种策略的核心思想是将修改频率高查询频率低的数据对象替换出去,选取修改频率低而查询频率又高的数据对象替换进来。LA2U 是将客户端对数据的访问次数与服务器对同一数据的更新次数的比值作为衡量标准,选取比值大的数据放入缓存替换掉比值较小的数据。

本文的主要内容是设计了一个缓存模型,模型包括缓存的粒度、缓存同步以及缓存替换三个部分。本文选择元组为缓存粒度,采用了基于消息摘要的同步算法,设计了一种基于数据时效性与事务优先级的缓存替换算法,并通过实验验证了该模型中算法的有效性。

1 缓存模型

如果一个移动设备的存储容量无限,那么就可以向移动客户端申请缓存所有的数据,显然在实际情况中,这是不现实的。缓存的粒度是缓存的基本问题,它可以优化缓存能力,因此,缓存粒度的选择对缓存模型有明显的影响。决定了缓存的粒度之后,下一步是要选择缓存同步算法,目的是要保持缓存的一致性。目前,已经有许多的缓存同步算法。移动客户端缓存命中率下降,客户端想添加一些新的数据缓存,此时就需要采用缓存替换算法来清除一些数据,为新的缓存数据腾出空间。据此,本文设计的缓存模型包括缓存粒度、缓存的同步以及缓存的替换。

1.1 缓存粒度

对于缓存粒度的选择,移动数据库系统中缓存的粒度可以是一行记录,也可以是属性。如果是以记录为缓存,需要传输整条记录的数据,此时缓存数据对部分用户来说是冗余的。而属性粒度是只存储用户频繁访问的属性数据,显然能够缓存更多的数据,有效利用有限的存储空间,网络开销相对来说较小,响应速度相对来说更快。其缺点是单一以属性作为粒度,缓存的命中率较低,且不利于同步。

以元组为缓存粒度时,移动客户端保存有元组中的每一个属性,这样的好处是能大大地提高缓存的命中率,而且,在移动客户端与服务器进行缓存同步的时候,将一行记录作为粒度更方便。因此,为了方便缓存同步,本文将元组,也即一行记录作为缓存粒度[11]。

1.2 同步算法

本文设计的缓存模型中采用文献[12]提出的基于消息摘要的缓存同步算法(Synchronization Algorithm based on Message Digest, SAMD)。该算法是采用比较消息摘要值的方式,通过两个消息摘要表中的标志值分析不一致的类型,然后使用主键来确定需要同步的那行数据。

基于消息摘要的算法将大量的数据压缩成一个消息摘要值,只需比较每行的消息摘要值,数据不一致性的检测得到简化,并最大限度地减少了存储空间的浪费。而且消息摘要函数的工作速度快,因此,对于计算能力较弱的移动客户端,负担也减少了。同步过程中,使用 SQL 批量处理查询移动客户端消息摘要表中的数据,查询结果通过网络发送到服务器端的数据库。这样,对移动客户端产生的负载是较小的,从而降低了网络访问所造成的开销。

但是该算法要维护一个单独的消息摘要表,从存储效率的角度来说,这是一个缺点。但 SAMD 算法不使用触发器、存储过程或时间戳。由于数据库设备厂商的独立性,SAMD 算法能用于服务器端数据库和移动数据库的任意组合。SAMD 只使用由国际标准化组织(International Organization for Standardization, ISO)的认证的标准的 SQL 语句,因此 SAMD 又具有可扩展性、适应性和灵活性等优点。

2 缓存替换算法

缓存模型中的第三个主要内容是缓存替换。缓存替换算法的作用是当空间已满需要腾出缓存空间时如何选择缓存中被替换的数据。对于缓存中的数据,哪些数据在断连时一定不会用到,哪些数据必须得留下,移动用户自身也不能完全确定。如果将缓存中被频繁访问的数据替换出去,会降低缓存的命中率,影响缓存的性能。

2.1 算法设计与分析

本文定义数据 d_i 的外部有效期 $t_{out}(i)$,用 $|t_{out}(i)|$ 表示其长度,设定两个值 t_1、t_2,当 $|t_{out}(i)| < t_1$ 数据 d_i 是高易变的;当 $|t_{out}(i)| > t_2$,数据 d_i 是稳定的;当 $t_1 \leqslant |t_{out}(i)| \leqslant t_2$,数据 d_i 是高易变的。

其他符号说明:C_s 表示移动客户端剩余缓存空间;v 表示缓存验证延迟;s_i 表示数据 d_i 的大小;p_i 表示数据 d_i 访问概率;r_i 表示从服务器获取数据 d_i 的代价。

根据以上定义,设计一个替换代价函数 $COST_i$:

$$COST_i = C_{replace}(i)/s_i \tag{1}$$

其中 $C_{replace}(i) = p_i * (|t_{out}(i)| * r_i + v)$,即:

$$COST_i = \frac{p_i * (|t_{out}(i)| * r_i + v)}{s_i} \tag{2}$$

前文已经假定移动客户端的缓存空间为 C_s,当 $C_s \neq 0$ 时,表示还有剩余缓存空间,缓存同步更新后新的缓存数据可以直接缓存;当 $C_s = 0$ 时,表示移动客户端缓存空间已满,要进行缓存替换,为新的缓存数据腾出空间。根据式(2),函数值越大,缓存数据 d_i 的价值越大,不应该被替换,本算法应该选取那些值越小的,即要达到以下目标:

1) $\sum_{d_i} COST_i$ 取得最小值;

2)

$$C_s + \lambda * \sum_{d_i \in C} s_i \geqslant \sum_{d_i \in P} Size(P) \tag{3}$$

其中:候选的新的缓存数据子集记为 P,$P = \{d_i \mid 1 \leqslant i \leqslant m\}$,$P \subseteq DB$;$Size(P)$ 表示新缓存数据的大小,将缓存数据按 $COST_i$ 的值排序,值越低排在越上面。

将每个数据 d_i 的时间特性定义为一个二元组 $(t_{out}(i), t_{get}(i))$,其中 $t_{get}(i)$ 是进入缓存的时间,定义 τ_c 为移动客户端缓存当前的时间,如果 $\tau_c - t_{get}(i) \leqslant t_{out}(i)$,则认为数据 d_i 是有效的;否则数据 d_i 是过期的。在移动客户端,缓存中数据的访问都是通过事务的调度进行的。而事务的执行通常是基于优先级。因此,本文定义 $pri(i)$ 表示事务的优先级,优先级越低的优先被替换。根据这些分析,本文设计一个选择函数:

$$SELECT(i) = \frac{\lambda |t_{out}(i)| + \alpha_i f(\tau_c)}{pri(i)} \tag{4}$$

其中 $\alpha_i (i = 1,2,3)$ 为加权因子,根据排好序的缓存数据,再利用式(4)获取缓存替换数据集,要满足选择函数取得最小

值$f(\tau_c)$:

$$f(\tau_c)=\begin{cases}\dfrac{e^{\tau_c-t_{get}(i)}}{t_{get}(i)-\tau_c}, & t_{get}(i)\neq\tau_c\\ 0, & t_{get}(i)=\tau_c\end{cases}\quad(5)$$

当数据 d_i 进入缓存的时间就是系统当前时间时,$f(\tau_c)$ 的值为 0;当数据 d_i 进入缓存的时间不是系统当前时间,$f(\tau_c)$ 的值如式(5),数据 d_i 进入缓存的时间越早,$f(\tau_c)$ 的值越小。

已有的缓存替换算法大都是通过考虑数据的访问频率、更新频率和数据的大小等特性来设计的,但是都没有考虑到数据的时效性以及事务的优先级。在移动客户端中,数据的实时性使得缓存数据只会在一定时间内流行,数据的值随着缓存同步更新后频繁地改变,因此,本文设计的缓存替换算法在考虑数据的大小、访问频率和更行频率的基础上,通过定义数据的有效期 $t_{out}(i)$,将数据的时效性考虑进来,又定义 $pri(i)$ 表示事务的优先级,综合考虑了替换的要素,这有助于提高缓存的命中率和事务成功率。

2.2　算法描述

符号定义　服务器中数据集合记为 DB;当前移动客户端缓存数据子集记为 C,$C=\{d_i\mid 1\leqslant i\leqslant n\}$,$C\subseteq DB$;缓存中选出来的进行替换的数据集合记为 R,$R\subseteq C$;q_{heap} 表示存放 MC(Mobile Client)缓存数据 id 的堆栈,最近使用越少的数据越接近栈顶。

缓存替换的伪代码如下:

```
Procedure replacement():
  get the new d_i from the Server;
                          //同步更新后有新的缓存数据要插入
  if C_s!=0                 //如果移动客户端缓存空间有剩余
  then insert the new d_i into the MC cache;
                                        //将新的缓存数据插入
  else
    find C.minimum(COST_i);   //缓存剩余空间不够,执行价值函数
  adjust the position of d_i in q_heap;
                        //根据价值函数将堆栈中缓存数据排序
  find C.minimum (SELECT(i));              //执行选择函数
  while C_s + λ * Σ_{d_i∈C} s_i ≥ Σ_{d_i∈P} Size(P) do
  remove the top item from the heap;
  clear the related from the MC cache;
                        //如果缓存的空间还不够,则循环执行移除
```

3　评价与分析

3.1　实验环境

实验采用局域网环境来模拟无线环境,用数据间隔的发送来实现对链路带宽的控制。实验局域网中有一台服务器,一台 MSS(Mobile Support Station)以及一台客户机,在客户机上创建多个线程来模拟多个客户端,每个客户端在内存中都建立自己的缓存区。具体实验环境的软硬件平台配置如表 1 所示。

表 1　系统实验的软硬件环境

角色	CPU	内存	系统
服务器	AMD M120 2.1 GHz	1 GB	Windows 2000 Serve
MSS	AMD M120 2.1 GHz	1 GB	Windows 2000 Server
客户端	Intel E2180 2 GHz	1 GB	Windows 2000 Server

本文所设计的缓存替换算法考虑了数据 d_i 的外部有效期 $t_{out}(i)$ 这一参数。在实验的时候,要对这一参数进行估算,本文采用滑动窗口值的方法。

所谓滑动窗口的方法,是在移动客户端缓存中,对数据 d_i 保持 ω 个近期更新的时间戳,用$\{t_i^1,t_i^2,\cdots,t_i^\omega\}$ 表示,因此,本算法采用式(6) 来表示缓存中数据 d_i 的外部有效期 $t_{out}(i)$:

$$t_{out}(i)=\frac{\omega}{t_c-t_i^\omega}\quad(6)$$

其中 t_c 是当前滑动时间窗口中数据 d_i 最早更新的时间戳。

移动客户端缓存中都有一个事务集,要尽量满足事务在到达截止期之前执行完毕。为了保证高优先级的事务优先被调度,本文采用可达截止期最早最优先(Earlist Feasible Deadline First, EFDF)。如果当前所有事务都具有不可达截止期,则按照截止期最早最优先(Earlist Deadline First, EDF)执行。

移动客户端对数据的访问与齐普夫定律类似,该定律表明,将移动客户端的缓存中的数据 d_i 按用户对它的访问概率进行排序,那么,对于任意一个数据 d_i 的访问概率与它的排名的乘积约为常数,而该数据 d_i 的访问概率与它的排名呈反比。因此,对照此模型,移动客户端对缓存中的数据 d_i 的访问概率表示为:

$$p_{di}=1\Big/\left(i^\theta\sum_{i=1}^{n}\frac{1}{i^\theta}\right);\quad 0\leqslant\theta\leqslant 1\quad(7)$$

其中:当 $\theta=0$ 符合正态分布,而当 $\theta=1$ 时符合严格的齐普夫分布;i^θ 是访问概率的排位。实验参数设置如表 2。

表 2　实验参数设置

参数	数值	参数	数值
MC 个数	1	$\lvert t_{out}(i)\rvert$	100 s
d_i 的个数	1 000	C_s	25% MC database size
事务集个数	80 000	$\min(d_i)$	1 KB
事务长度	8	$\max(d_i)$	100 KB
数据访问	齐普夫分布	事务优先规则	EFDF,EDF

3.2　实验结果与分析

本文将所设计的替换算法与现在常用的 LRU、LA2U 算法进行比较,主要从缓存命中率和事务重启率两个方面对缓存替换算法进行性能评价,得到的实验结果如图 1 ~2。

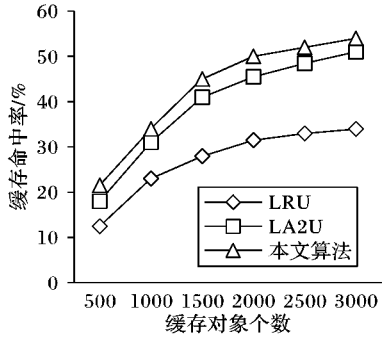

图 1　3 种算法缓存命中率比较

根据图 1,随着缓存数据个数的增多,3 种缓存替换算法的缓存命中率都有提高,本文提出的缓存算法命中率最高。这是因为 LRU 是将缓存中最近最少使用的数据替换出移动客户端;LA2U 替换算法是将更新频率高访问频率低的数据对象替换出去,它是将客户端对数据的访问次数与服务器对同一数据的更新次数的比值作为衡量标准,选取比值大的数据放入缓存,替换掉比值较小的数据;而本文提出的算法,不仅将访问频率和更新频率纳入考虑,还综合了数据的外部有

效期,数据的流行性以及事务的优先级,更全面,因此,命中率相对来说更高。

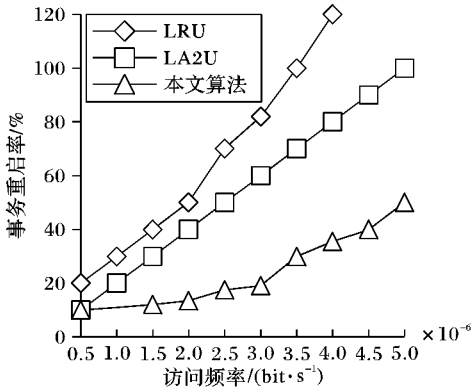

图 2 3 种算法的事务重启率比较

根据图 2,事务的重启指的是移动用户对缓存进行操作,提交的事务中相关联的数据过期导致事务提交失败被迫重启,显然,LRU 和 LA2U 都没有考虑到数据的有效期与事务的优先级,因此,事务重启率比本文提出的算法要高。

4 结语

本文针对移动数据库系统中经常断连的问题,设计了缓存模型,对缓存模型中的几个关键问题进行了分析与探讨,包括缓存的粒度、缓存同步、缓存的替换。将元组作为缓存粒度,能大大地提高缓存的命中率。基于消息摘要的同步策略通过比较移动客户端与服务器消息摘要表中消息摘要值,判断需要同步的行,再根据消息摘要表中标志位的比较,判断不一致的类型,从而完成缓存同步,维护移动客户端缓存与服务器数据的一致性。考虑到数据的时效性与事务的优先级,设计了一种基于价值函数的缓存替换算法。在该算法中,不仅考虑移动用户对缓存数据的访问频率和数据的大小,而且在价值函数中将数据的外部有效期纳入考虑,利用价值函数将缓存数据排序,值最小的排在最上面;再根据事务的优先级设计一个选择函数,选择函数值最低的数据作为替换数据。使用该算法能够提高缓存的命中率,同时可以降低事务的重启率,但是网络开销略有增加。

参考文献:

[1] IMIELINSKI T, BADRINATH B R. Mobile wireless computing: challenges in data management [J]. Communications of the ACM, 2005, 37(10): 18 - 28.

[2] KUMAR A, SARJE A K, MANOJ M. Prioritised predicted region based cache replacement policy for location dependent data in mobile environment [J]. International Journal for Ad Hoc and Ubiquitous Computing, 2010, 5(1): 56 - 67.

[3] MADHUKAR A, OZYER T, ALHAJJ R. Dynamic cache invalidation scheme for wireless mobile environments [J]. Wireless Networks, 2009, 15(6): 727 - 740.

[4] ANANDHARAJ G, ANITHA R. A distributed cache management architecture for mobile computing environments [C]// IACC 2009: Proceedings of the 2009 IEEE International Advance Computing Conference. Piscataway: IEEE Press, 2009: 642 - 648.

[5] GRAY J, HELAND P, ONEIL P, *et al*. The danger if replication and a solution [C]// Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1996: 173 - 182.

[6] WOLFSON O, JAODIA S, HUANG Y X. An adaptive data replication algorithm [J]. ACM Transactions on Database Systems, 1997, 22(2): 255 - 314.

[7] 陈舒, 姜宁康. 基于多方面完整性检测的移动数据同步机制[J]. 计算机应用, 2009, 29(6): 184 - 188.

[8] ACHARYA S, FRANKLIN M, ZDONIK S. Dissemination-based data delivery using broadcast disks [J]. IEEE Personal Communications, 2007, 2(6): 122 - 127.

[9] 李霖. 移动数据库中数据广播与复制/缓存技术的研究[D]. 长沙: 国防科学技术大学, 1998.

[10] ANANDHARHJ G, ANITHA R. A power-aware low-latency cache management architecture for mobile computing environments [J]. International Journal of Computer Science and Network Science, 2008, 8(10): 121 - 126.

[11] 韩向春, 边玮, 沈峰, 等. 代理缓存替换一致性算法的研究[J]. 计算机工程与设计, 2009, 30(11): 2734 - 2736.

[12] CHOI M Y, CHO E A, PARK D H, *et al*. A database synchronization algorithm for mobile devices [J]. IEEE Transactions on Consumer Electronics, 2010, 56(2): 392 - 398.

(上接第 3219 页)

[2] GARDNER G Y. Visual simulation of clouds [J]. Computer Graphics, 1985, 19(3): 279 - 303.

[3] NISHITA T, SIRAI T, TADAMURA K, *et al*. Display of the earth taking into account atmospheric scattering [C]// Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques. New York: ACM Press, 1993: 175 - 182.

[4] DOBASHI Y, NISHITA T, OKITA T. Animation of clouds using cellular automaton [C]// Proceedings of Computer Graphics and Imaging'98. New York: ACM Press, 1998: 251 - 256.

[5] STAM J. Stable fluids [C]// Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques. New York: ACM Press, 1999: 121 - 128.

[6] STAM J, FIUME E. Depicting fire and other gaseous phenomena using diffusion processes [C]// Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques. New York: ACM Press, 1995: 129 - 136.

[7] MIYAZAKI R Y O, DOBASHI Y, NISHITA T. An efficient cloud visual simulation using adaptive grid method [J]. IEICE Transactions on Information and Systems, 2004, 87(9): 1814 - 1822.

[8] 徐江斌, 赵健, 杨超, 等. 真实感云的快速建模[J]. 小型微型计算机系统, 2010, 31(8): 1590 - 1594.

[9] 吴银霞, 陈雷霆, 何明耘. OpenGL 中基于粒子系统雷达扫描实时模拟[J]. 计算机应用, 2009, 29(1): 258 - 260.

[10] 张芹, 张健, 闵建平. 提高粒子系统实时性的方法研究[J]. 计算机工程, 2003, 29(18): 46 - 48.

[11] DOBASHI Y, KANEDA K, Y AMASHITA H, *et al*. A simple, efficient method for realistic animation of clouds [C]// Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. New York: ACM Press, 2000: 19 - 28.

[12] HARRIS M J. Real-time cloud simulation and rendering [D]. Wilmington: University of North Carolina, 2003.

[13] 何晓曦, 陈雷霆, 朱清新. 一种简化的流体方法快速仿真大型三维云场景[J]. 计算机应用研究, 2012, 29(6): 2357 - 2359.

[14] 姜洪洲, 黄会林. 视景系统云层的特效模拟[J]. 计算机仿真, 2007, 24(7): 225 - 228.

[15] 任威, 梁晓辉, 马上, 等. 大规模三维云实时模拟方法[J]. 计算机辅助设计与图形学学报, 2010, 22(4): 662 - 669.