一种运用块级局部性的闪存缓存管理策略①

龚剑峰[1], 李 曦[1,2], 陈香兰[1], 朱宗卫[1], 贾刚勇[1]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: 闪存被广泛应用在电子产品的存储设备中, 针对闪存的研究也日益得到重视. 基于访问的局部性原理, 并结合闪存读写代价的差异性, 提出了一种针对闪存特点运用块级局部性原理的 cache 缓存管理算法 LRU-BLL. 实验表明, 这种方法有效地提高了缓存的命中率, 并且减少了缓存的脏页回写次数和提高了缓冲区的平均换出长度.

关键词: 闪存; 缓存管理; 擦写; 局部性

Block Level Locality Aware LRU Cache Management Strategy for Flash Memory Storage

GONG Jian-Feng[1], LI Xi[1,2], CHEN Xiang-Lan[1], ZHU Zong-Wei[1], JIA Gang-Yong[1]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Suzhou Institute for Advanced Study, University of Technology and Science of China, Suzhou 215123, China)

Abstract: Flash memory is widely used as storage media in electronic devices recently, and the researches of flash memory also receives more and more attention. This paper proposes a cache management algorithm called LRU-BLL, which is based on block level locality and replacement cost difference of clean page and dirty page, and combines with character of flash memory. By using simulation to evaluate the scheme, we find hit ratio and number of write/erase operations have a satisfactory improvement in proposed algorithm.

Key words: Flash memory; cache management; write and erase; locality

1 引言

闪存(flash memory)1984 年由东芝公司提出[1]. 相比于传统硬盘, 闪存具有低功耗、尺寸小、抗震性好的优势[2]. 闪存是一种特殊的 EEPROM, 主要分为 Nand flash 和 Nor flash 两种, 具有“写前擦除”和“有限擦除次数”两个特点. 闪存中的数据只能从 1 改成 0, 从 0 改成 1 需要通过擦除操作来实现, 即写前擦除. 这种擦除操作是以块为单位执行的, 而读写以页为单位执行. 修改闪存上的数据, 一种方法是先在内存中修改对应的块, 然后擦除闪存上的原数据块, 再将内存中含有最新数据的块写入到闪存中原位置, 称为本地更新; 另一种方法是将新数据块写入到闪存中的空闲块中, 称为异位更新. 因为本地更新导致块的擦写不能均衡分布, 所以闪存中使用的都是异位更新方式.

此外, 闪存块的可擦写次数有限, 当超过这个上限时, 数据的有效性就不能得到保证.

目前国内外对闪存的研究多集中在减少擦写次数方面. 通过缓存管理策略可以改善闪存的擦写次数和归并代价, 本文提出一种新的缓存管理算法 LRU-BLL(LRU concern for block level locality), 利用块级局部性, 延缓脏页回写, 并兼顾缓冲区的换出长度. 论文的第 2 部分分析闪存性能优化领域的相关工作和存在的问题, 第 3 部分说明 LRU-BLL 算法的实现细节, 第 4 部分是实验和结果的分析, 第 5 部分进行总结.

2 相关工作

基于闪存的存储系统如图 1 所示. 闪存被分为数据区(Data block area)和缓冲区(Log buffer)两部分, 它

① 基金项目:国家自然科学基金(61272131,61202053);江苏省自然科学基金(SBK201240198);江苏江苏省产学研前瞻性联合研究项目(BY2009128)资助.
收稿时间:2012-12-29;收到修改稿时间:2013-01-28

们均由多个数据块组成, 每个块中含有一定数量的页. 数据区是数据的实际存储区域, 缓冲区用于缓冲回写的脏页, 以降低脏页回写的代价. 缓冲区含有固定数量的块[3], 数据区的数据都首先被更新在此处, 然后再和原数据块归并得到新的数据块. FTL 层(Flash Translation Layer)负责闪存的地址映射, 对外模拟成普通的块设备. Cache 是内存中的数据缓存. 当缓存空间不足时, 换出的脏页会被保存到下层缓冲区中.

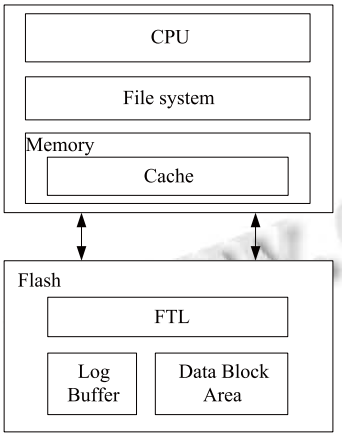


图 1　基于闪存的存储系统

缓冲区和数据区有 BAST 和 FAST[3,4]两种关联方式. 对于 BAST, 缓冲区中每个块仅包含一个数据块的页; 而 FAST 方式中, 缓冲区的每个块可包含多个数据块的页. 当缓冲区被填满或者其中的某块被写满时, 需要回收缓冲区和数据区的对应块, 称为归并. 首先说明, log block 表示缓冲区的块而 data block 表示数据区的块, 如图 2 所示, BAST 有三种归并方式: 交换归并(switch merge)、部分归并(partial merge)和全归并(full merge), 实际采用的归并方式需要根据 log block 的状态来选择.

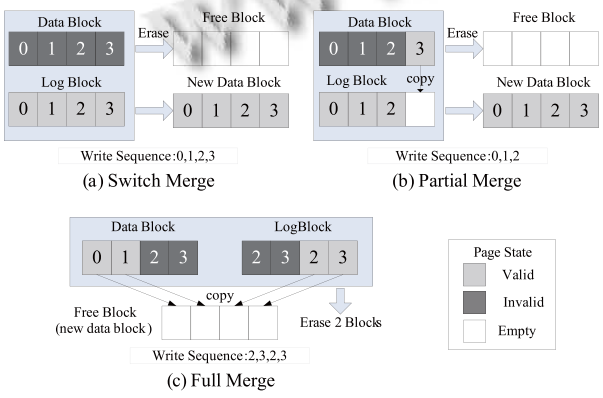


图 2　BAST 方式的 3 种归并操作

假设块中含有 n 页, 交换归并将 log block 作为新的数据块, 原数据块被擦除成为空闲块, 需要 1 次擦除; 部分归并将原数据块的页拷贝到 log block 中, 然后原数据块被擦除成空闲块而 log block 作为新的数据块, 需要 1 次擦除和小于 n 次的拷贝; 而全归并将原数据块和 log block 的数据拷贝到一个空闲块中形成新的数据块, 然后两者均被擦除成空闲块, 需要 2 次擦除和 n 次拷贝. 可见代价高昂的全归并操作需要尽可能避免.

归并时 log block 中有效页的个数称为缓冲区的“换出长度”. 提高归并时缓冲区的平均换出长度, 有利于提高交换归并出现的概率, 并同时可减少归并时需要的拷贝次数, 从而降低闪存整体上的归并代价[5].

缓存管理的代表性算法有 CFLRU、LRU-WSR 和 FAB[6-8]等.

缓存在执行换出操作选取候选页时, 传统的 LRU 算法并不考虑页的干净与否, 而是简单地选取 LRU 端的页作为候选页. 因为在缓存中换出脏页的代价远比干净页高, 所以需要减少脏页的换出次数. CFLRU 算法通过延长脏页在缓存中的滞留时间来减少脏页的换出次数.

如图 3 所示, Working Region 存放最近使用的页, 而 clean-first region 存放准备被换出的页. 当需要换出缓存中的页时, 首先在 clean-first 区域寻找干净页换出; 如果此区域没有干净页, 那么换出 LRU 端的脏页. 对于 CFLRU 算法, 有一些改进方案如 CFLRU/C, CFLRU/E, DL-CFLRU/E[9]等, 试图进一步降低换出代价.

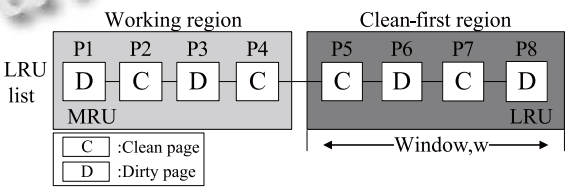


图 3　缓存管理算法 CFLRU 示例

LRU-WSR 算法如图 4 所示, 也是基于缓存中换出干净页和脏页代价的不对称性, 但是它进一步对脏页的冷热进行区分, 优先换出干净页和冷脏页, 延迟换出缓存中访问较频繁的热脏页, 进一步减少脏页的换出次数从而减少擦写操作, 降低闪存访问开销. 基于 LRU-WSR 算法的改进还有 CCFLRU[10], 它更进一步的区分干净页的冷热属性.

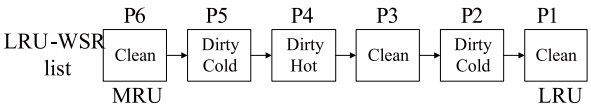


图 4 缓存管理算法 LRU-WSR 示例

FAB 算法如图 5 所示, 它将页按块分组, 组织在 LRU 链表中, 当缓存空间不足时, 选取含有最多页的块并将其全部换出. FAB 算法的目的在于提高缓冲区的平均换出长度, 因此 FTL 可以更多地使用交换归并或者部分归并来回写数据, 避免了代价高昂的全归并操作, 达到降低数据归并代价的目的[5].

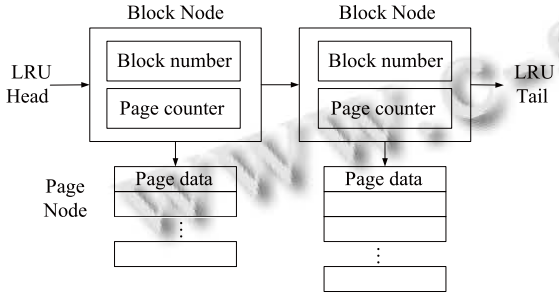


图 5 缓存管理算法 FAB 示例

此外, FOR 算法[11]根据读写的代价和热度为缓存中的每个页赋予权重, 在执行换出时以这个权重作为依据换出页. GDLRU 算法[12]将缓存页分成两组, 干净页链表和脏页链表, 优先换出干净页, 当不存在干净页时再换出脏页, 以减少脏页回写的次数. 而 AD-LRU 算法[13]考虑访问频率, 将缓存分为自适应的冷区和热区, 两个区域中都使用 LRU 算法, 其中冷区存放只被访问过一次的页且会被优先替换, 而热区存放被多次访问的页.

这些缓存管理算法, 或者考虑脏页换出的代价, 延缓脏页的换出, 但却未考虑降低数据归并代价; 或者通过将相同块内的页一次性换出, 降低后续归并的代价, 但却影响了缓存的命中率.

CFLRU 和 LRU-WSR 算法没有充分利用块级局部性, 即如果某页被访问, 那么该数据块内的其他页很可能也即将被访问. 而 CFLRU 和 LRU-WSR 因为不考虑这些, 所以会出现即将被访问的页被换出的情况, 降低命中率. 并且, 在遇到顺序写序列时, 因为没有考虑相同块内的页间关系, CFLRU 和 LRU-WSR 算法会退化成 FIFO 算法. 这时缓存中相同块内的脏页未能聚簇在一起并连续换出, 导致同一个块中的不同脏页被间歇性地替换出去, 这样可能仅仅因为更新两个页的数据而产生两次数据块的归并, 即产生了震荡, 导致额外的读写、擦除开销, 影响性能和能耗.

另外, FAB 算法以块为单位换出, 本意是希望通过一次性换出整个块中的页来提高缓冲区的平均换出长度, 但这样会使缓存中的页面数变少, 从而降低了缓存的命中率.

3 LRU-BLL算法

本文提出一种新的缓存管理算法 LRU-BLL(LRU concern for block level locality), 该算法考虑了访问的块级局部性, 在缓存中注重同块页间的聚簇, 同时延缓脏页的回写.

如图 6 所示, 缓存中的页按照所属块的不同被分配到不同的组, 各个组组成 LRU-BLL 链表保存在缓存中. 每个组中保存着近期被访问的页, 组中的页按照 LRU 顺序排列, 同时, 组之间也按照 LRU 的顺序排列, 最新被访问的组放置在 LRU 链表的 MRU 端. 链表 LRU 端的组称为候选块, 当缓存需要新的空间时, 从候选块中换出页.

缓存页命中时, 更新命中页在组中的位置, 并将对应组移到 MRU 端. 缓存失效时, 如果缓存有剩余空间, 将该页加入所在组(没有则创建)并将整个组迁移到 MRU 端; 缓存无剩余空间时, 选取 LRU 端的组作为候选块, 换出候选块中的一页, 然后再将读取的失效页添加进对应的组并将整个组移至 MRU 端. 换出候选块中的页时, 优先换出其中的干净页, 无干净页时, 再换出候选块中的脏页.

当候选块所属块中的页被再次访问, 即该组所属块被再次访问时, 停止该候选块的换出并将其移到 MRU 端, 暂停对候选块中页的聚簇换出, 然后重新选取 LRU 端的组作为新的候选块. 当下次因为缓存失效并且需要换出页时, 换出新候选块中的页.

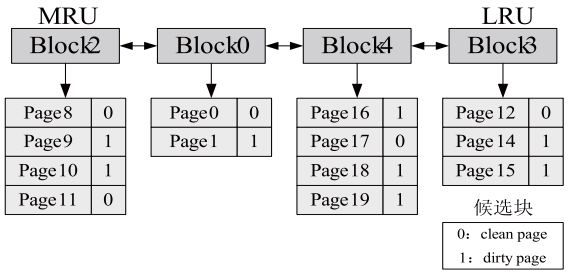


图 6 缓存管理算法 LRU-BLL 示例

根据局部性原理, 如果将 LRU 端的候选块依次以页为单位换出, 那么被连续换出的多个页属于同一个块, 而缓存中被连续换出的页会在缓冲区中累积, 这样就提高了缓冲区的平均换出长度, 使 FTL 层更多的采用交换归并和部分归并方式回写脏页, 从而降低数据归并的代价. 并且当候选块中的页被再次访问时, 候选块暂停换出并被移至 MRU 端, 这就避免了即将被访问的页被换出缓存, 从而提升了缓存的命中率.

4　实验与结果分析

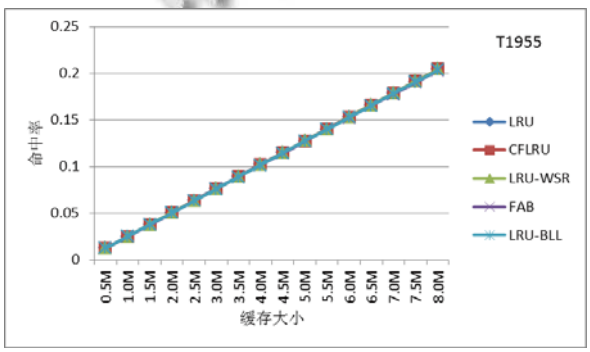
本文采用表 1 中的测试用例[9]来模拟对闪存的访问操作, 每个测试用例的特性如表中所示. 如 T9182 有 300000 次读写操作, 其中 90%是读操作, 10%是写操作, 并且 80%的操作集中在 20%的区域. 假定每个块是 64 页, 每页 4K, log buffer 中含有 2 个数据块, 且采用 BAST 算法[3], 而缓存采用 LRU、CFLRU、LRU-WSR、FAB 和 LRU-BLL 算法, 缓存从 0 到 8M 变化. 通过模拟实验[9], 采集了以下性能参数: 缓存的命中率和脏页回写次数, 以及缓冲区的平均回写长度. 实验结果表明 LRU-BLL 算法提高了缓存命中率, 减少了缓存中的脏页回写次数, 提高了缓冲区的平均换出长度.

表 1　实验测试用例的特性

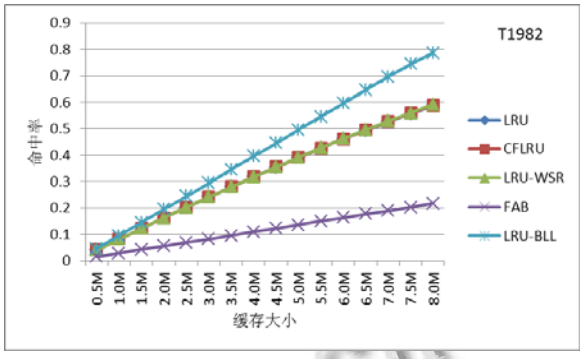
类型	访问次数	访问的不同数据页个数	读写比例	访问局部性
T9182	300000	10000	90%/10%	80%/20%
T5582	300000	10000	50%/50%	80%/20%
T1982	300000	10000	10%/90%	80%/20%
T9155	300000	10000	90%/10%	50%/50%
T5555	300000	10000	50%/50%	50%/50%
T1955	300000	10000	10%/90%	50%/50%

4.1 命中率

图 7 中可以发现, 访问序列局部性不同时, 算法表现出了不同的效果.



(a)　用例 T1955

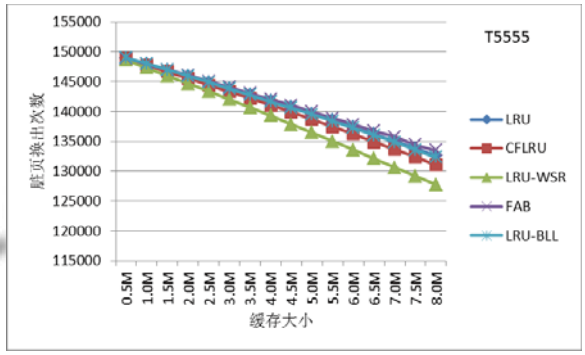


(b)　用例 T1982

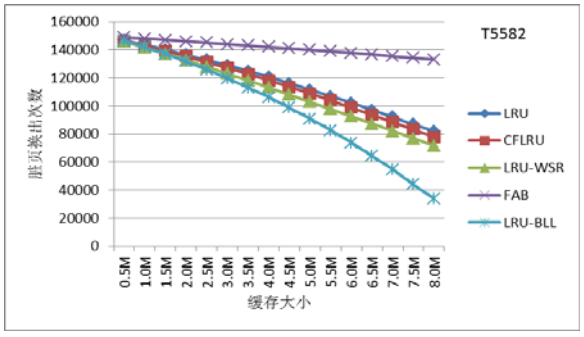
图 7　缓存命中率比较

局部性较差时(如 T1955), 整个过程可以视为一种顺序访问, 所以这些使用局部性的缓存算法都会失去效果, 退化成类于 FIFO 的先进先出算法. 而局部性较好时(如 T1982), LRU-BLL 算法在考虑对脏页聚簇的同时还考虑了块级访问局部性, 即当某块中的页被访问时, 该块的其他页也可能会很快被访问, 所以有效提高了缓存的命中率, 而 CFLRU 和 LRU-WSR 因为没有考虑这一点, 所以效果基本类似于 LRU 算法.

4.2 Dirty page



(a)　用例 T5555



(b)　用例 T5582

图 8　缓存脏页回写次数的比较

如图 8 所示, 局部性较差时(如 T5555), CFLRU 和 LRU-WSR 呈现出优势, 因为这两种算法在更大的区域范围内强调脏页的延迟回写, 而 LRU-BLL 会在换出候选块中的干净页后陆续换出该组中的脏页, 所以 CFLRU 和 LRU-WSR 会比 LRU-BLL 换出更少的脏页, 从而有更少的脏页回写次数. 局部性较好时(如 T5582), 因为 LRU-BLL 在候选块内考虑干净页和脏页的换出代价差异, 优先换出组内的干净页. 并且因为访问的块级局部性, 当候选块被再次访问时, 该组中的脏页会在换出前被移至 MRU 端, 所以脏页的换出得到有效延缓, 因此较 LRU、CFLRU 和 LRU-WSR 算法, LRU-BLL 有更少的脏页换出次数. FAB 算法以块为单位换出其中的页, 且未考虑脏页的延迟回写, 所以脏页回写次数最高.

4.3 平均回写长度

从图 9 中可以发现, 相比于 LRU、CFLRU 和 LRU-WSR 算法, FAB 和 LRU-BLL 关注了页的块属性, 在缓存中聚合相同块内的页, 这样在被换出到缓冲区中时, 因为连续换出页间聚合度的提高, 所以缓冲区中的平均块长度也会得到提高. 由于 LRU-BLL 考虑了块级局部性, 认为某块中的页如果被访问, 那么该块中的其他页后续很可能也会被访问, 这样候选块在换出过程中会因为再次被访问而停止换出剩余的页, 转而选出新的候选块, 所以一定程度上破坏了换出页间的聚合度, 因此 FAB 算法的缓冲区换出长度提高效果最好.

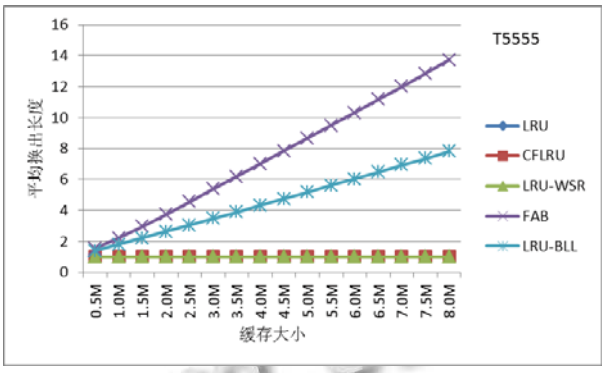
综上, 虽然 LRU-BLL 的平均回写长度小于 FAB, 但是其命中率以及脏页回写次数都大大优于 FAB 算法. LRU-BLL 算法很好的融合了 LRU 的命中率, CFLRU、LRU-WSR 的脏页回写延迟, 以及 FAB 的聚合度, 有效地提高了闪存的性能.

5 结语

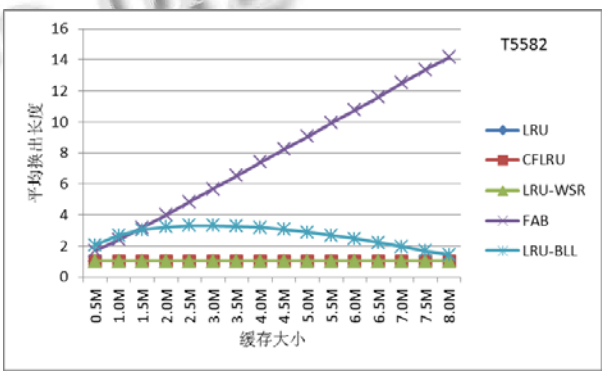
根据闪存的自身特点, 并结合局部性原理在闪存块中的应用, 论文提出了 LRU-BLL 算法, 该算法的贡献有几点:

1) 提高缓存命中率. 根据块级局部性, 将被访问块中的其他页移到 MRU 端, 避免这些即将被访问的页因缓存空间不足被换出, 提高缓存的命中率.

2) 减少脏页的回写次数. 缓存空间不足需要换出页的时候, 优先换出候选块中的干净页, 延缓脏页的换出, 减少了脏页的换出和回写次数, 这对闪存的性能和寿命大有帮助.



(a) 用例 T5555



(b) 用例 T5582

图 9 缓冲区平均回写长度的比较

3) 提高缓冲区的平均换出长度. 以页为单位连续换出相同块内的页, 可以有效提高页的聚簇程度, 避免因为间歇换出的脏页属于同一个块而可能引起的不必要数据更新.

实验表明 LRU-BLL 算法有效地提升了缓存的效果, 缓存的命中率和脏页的换出次数得到改善, 并同时提高了缓冲区的平均换出长度.

参考文献

1 Masuoka F, Asano M, Iwahashi H, Komuro T, Tanaka S. A new Flash E2PROM cell using triple polysilicon technology. IEDM Tech. Dig, 1984: 464–467.

2 Douglis F, Cáceres R, Kaashoek F, Li K, Marsh B, Tauber JA. Storage alternatives for mobile computers. Proc. of the 1st Symposium on Operating Systems Design and Implementation Monterey. CA: USENIX, 1994: 25–37.

3 Kim J, Kim JM, Noh SH, Min SL, Cho Y. A Space-Efficient Flash Translation Layer for Compact-Flash Systems. IEEE

Trans. on Consumer Electronics, 2002,48(2):366–375.
4 Lee SW, Park DJ, Chung TS, Lee DH, Park S, Song HJ. A Log Buffer-based Flash Translation Layer Using Fully Associative Sector Translation. ACM Trans. on Embedded Computing Systems, 2007,6(3): 1–27.
5 Kang S, Park S, Jung H, Shim H, Cha J. Performance trade-offs in using NVRAM write buffer for Flash memory-based storage devices. IEEE Trans. on Computers, 2009,58(6): 744–758.
6 Park SY, Jung D, Kang J, Kim J, Lee J. CFLRU: a Replacement Algorithm for Flash Memory. International Conference on Compilers, Architecture and Synthesis for Embedded Systems(CASES). 2006: 234–241.
7 Jung H, Shim H, Park S, Kang S, Cha J. LRU-WSR: Integration of LRU and writes sequence reordering for flash memory. IEEE Trans. on Consumer Electronics, 2008,54(3): 1215–1223.
8 Jo H, Kang JU, Park SY, Kim JS, Lee J. FAB: flash-aware buffer management policy for portable media players. IEEE Trans. on Consumer Electronics, 2006,52(2):485–493.
9 Yoo YS, Lee H, Ryu Y, Bahn H. Page replacement algorithms for NAND flash memory storages. International Conference on Computational Science and Its Applications. 2007, 1: 201–212.
10 Li Z, Jin P, Su X, Cui K, Yue L. CCF-LRU: A new buffer replacement algorithm for flash memory. IEEE Trans. on Consumer Electronics, 2009,55(3):1351–1359.
11 Lv Y, Cui B, He B, Chen X. Operation-aware buffer management in flash-based systems. Proc. of SIGMOD Conference. 2011: 13–24.
12 Lin MW, Chen SY, Wang GP. Greedy Page Replacement Algorithm for Flash-aware Swap System. IEEE Trans. on Consumer Electronics, 2012,58(2):435–440.
13 Jin P, Ou Y, Härder T, Li Z. AD-LRU: An efficient buffer replacement algorithm for flash-based databases. Proc. of Data Knowl. Eng. 2012: 83–102.

(上接第 143 页)

本系统的组件包括: 投诉页面; 投诉分配页面; 投诉处理页面; 系统管理页面; 登录页面; 投诉者投诉程序; 投诉分配程序; 投诉处理程序; 投诉管理程序; 系统信息维护程序; 数据管理程序. 系统部署为 B/S 模式, 包括客户端、应用服务器和数据库服务器.

3 结语

通过汉中茶叶客户投诉管理系统的实例分析, 可以看出, UML 建模, 适用于以面向对象技术进行系统开发的不同阶段, 在支持面向对象开发方法增量迭代的开发原则下, 将软件系统用简单明了的可视化图形表示出来, 不仅有助于建立清晰、直观的系统模型, 还可提高系统的可重用性和维护性、降低了软件开发的复杂度, 也为程序员提供了统一的交流手段, 提高了软件开发效率, 具有广泛的应用前景.

参考文献

1 邱郁惠.系统分析师 UML 实务手册.北京:机械工业出版社, 2008.23–60.
2 王苹.基于 UML 建模的销售系统研究.煤炭技术,2011,6: 268–270.
3 鲍海琴,王振华,等.基于 UML 的电子商务系统分析与设计.电脑编程技巧与维护,2009,6:24–26.
4 夏克付,李心科.基于 UML 的电子商务系统建模研究.计算机与现代化,2009,6:30–33.
5 刁成嘉.UML 系统建模与分析设计.北京:机械工业出版社, 2009.45–52.