BWMMS 元数据分布信息缓存管理*)

杨德志[1,2] 许 鲁[1] 张建刚[1]
(中国科学院计算技术研究所 北京 100080)[1] (中国科学院研究生院 北京 100039)[2]

摘 要 BWMMS是BWFS的分布式文件系统元数据服务子系统。它充分利用系统访问负载的动态性和局部性特征,通过简单的集中决策机制管理元数据请求负载在多个元数据服务器的分布。为降低集中决策点可能的瓶颈限制,集中决策点位于元数据请求处理路径的末端。本文介绍各个元数据服务器上用来降低对后端集中决策点的压力,提高元数据访问效率的元数据分布信息缓存,并通过测试数据评估缓存命中率对后端集中决策点和元数据访问效率的影响。

关键词 分布式文件系统元数据服务,元数据分布信息管理,缓存有效性

Management of Metadata Distribution Information Cache in BWMMS

YANG De-Zhi[1,2] XU Lu[1] ZHANG Jian-Gang[1]
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)[1]
(Graduate School of the Chinese Academy of Sciences, Beijing 100039)[2]

Abstract As the subsystem of BWFS for distributed file system metadata service, BWMMS uses a simple "central decision, distributed process" mechanism and dynamic decision policy to manage distribution of file metadata workload among multiple metadata servers. To decrease requests needed to the backend distribution decision maker and to increase metadata request efficiency, a metadata distribution cache is introduced on each metadata servers. In this paper, we describe the organization and management of metadata distribution cache and its influence on the backend server and file system metadata efficiency in detail.

Keywords Distributed file system metadata service, Metadata distribution management, Cache efficiency

1 引言

分布式文件系统元数据访问请求数量比例超过50%[1],因此文件系统元数据服务成为分布式文件系统研究热点。服务器集群方式是可扩展元数据服务器主要系统结构,如何有效管理元数据请求负载在多个服务器的分布成为文件系统元数据服务的主要问题。

本文介绍蓝鲸多元数据服务器系统(BWMMS)的元数据分布管理机制和策略以及元数据服务器的分布信息缓存管理,并通过初步的实验数据评估元数据缓存对系统元数据性能的影响。

本文第2节介绍本研究的背景,第3节着重介绍元数据服务器上负责管理元数据分布信息缓存机制的设计和实现,第4节通过初步的实验数据验证BWFS的元数据分布信息缓存的影响,第5节介绍相关研究的情况,最后是总结及下一步的工作概况。

2 研究背景

BWFS[2]是国家高性能计算机工程技术研究中心(NRCHPC)自主设计和实现的基于IP协议的海量网络存储系统BW1K的分布式文件系统。它包括通过高速互联网络连接的系统用户(Application Server, AS)、文件系统元数据服务提供者(System Servers)、文件系统数据服务提供者(Storage Nodes)和系统管理服务提供者(Administrator)等四部分,其系统结构如图1所示。

在图1中,存储节点(storage node)为存储系统用户提供集中的数据存储服务,系统至少有一个存储节点。系统服务器(system servers)是提供分布式文件系统元数据服务的服务器集合,它至少包括一个元数据服务器。管理服务器(Administrator)是系统中唯一的监控服务器。系统的集中管理有助于以简单方便的管理方式降低系统的管理成本。client是在用户服务器上加载的软件模块。

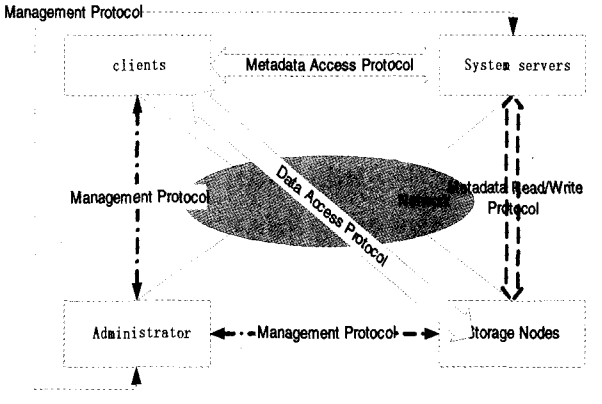

图1 BW1K系统结构

用户访问文件数据包括获得描述文件数据的元数据和获取实际文件数据两个过程。BWFS将用户文件访问的控制流和数据流分离开来,以提供并发高效的文件数据访问。

3 BWFS元数据服务

BWMMS是图1中的系统服务器部分,它以元数据服务器集群方式为系统用户提供分布式文件系统元数据服务。

3.1 BWMMS的服务器构成

*)国家自然科学基金项目"面向网络存储的集群系统的体系结构",受理号60373045。

系统主要有两种元数据请求负载分布决策机制。一种是分布式决策机制，各个元数据服务器独立收集决策需要的信息，然后在决策完成后将结果通知给其他所有的元数据服务器；另一种是集中决策机制。单一的集中决策服务器决定元数据请求负载的分布，决策结果不需要广播给所有的元数据服务器，需要知道决策结果的服务器从集中决策服务器获得相应的信息。相对于分布式决策而言，集中决策服务器虽然存在集中服务器成为系统瓶颈的可能，但是，集中决策机制能够简化系统服务器的交互，减轻提供元数据服务的元数据服务器的负担，并能够以简单的方式完成系统的具体实现。

BWMMS 采用集中决策机制。系统由大量的处理用户元数据请求的元数据服务器和决定元数据请求分布的绑定服务器构成。用户的元数据请求提交给元数据服务器。元数据服务器与后端的绑定服务器(BS)通信，明确请求的处理服务器。如果元数据请求处理过程 MS 必须和 BS 通信，BS 的负载将非常大，请求的处理时延也将增大。为降低 BS 的负载，缩短用户元数据请求处理时延，各个 MS 需要缓存 BS 的决策结果。只有在缓存信息无效时，MS 才会和 BS 通信，并用获得的信息刷新缓存。

3.2 BWMMS 元数据分布管理机制和策略

为支持 BWMMS 元数据服务器的扩展，简化系统的结构，BWMMS 采用层次化的元数据分布信息缓存来管理元数据分布信息，如图 2 所示。

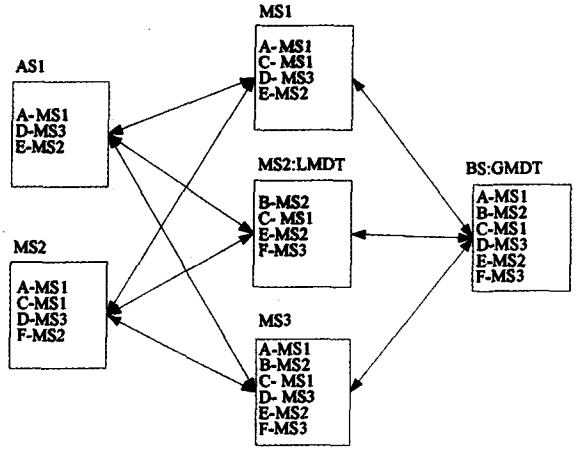

图 2 BWMMS 层次化元数据分布缓存机制

在层次化缓存结构的底层(图 2 的最右边)，BS 负责决定系统的元数据分布。根据已有研究的结论，系统中的数据在同一时间因为被访问而活跃比例非常小。所以，BS 仅针对活跃数据进行请求分布决策。这样它就可以管理更大规模的数据，表现出一定的数据管理的扩展能力。

在最上层的客户部分(图 2 的最左边)，客户缓存自己访问到的元数据的分布信息，这些信息的生命周期与相应的元数据一致。当元数据因为不活跃被释放时，其分布信息在客户机中的缓存也将消失。当元数据被再次访问时，客户首先需要通过访问某个元数据服务器获得该元数据最新的分布信息，然后再进行元数据的请求。

位于层次结构中间部分的是各个元数据服务器上的元数据分布信息缓存，也是本文后续部分的主要内容。它缓存分布在自己的全部活跃元数据分布信息，同时还缓存客户通过自己从 BS 获得的元数据分布信息。当 MS 上元数据因为没有被访问而不活跃时，它将释放自己缓存该元数据的分布信息，并通知 BS 释放对该元数据的分布决策。当 BS 释放掉分布信息后，该元数据彻底变成不活跃，不再受系统管理。当元数据因为被再次访问而变成活跃时，系统各个元数据分布信息缓存被填充，系统又开始管理该元数据的分布。

3.3 MS 的元数据缓存管理

在实现上，BS 以哈希串表方式组织内存，每个项的哈希串表通过 list = hash(ino#)决定，BS 中元数据分布信息项的生成和释放通过各个 MS 的访问驱动。

AS 通过扩展 Linux 的文件索引节点结构，直接将其记录在相应的字段中，分布信息的生存周期与文件索引节点一致，缓存管理比较简单。

本节主要介绍 MS 元数据分布信息缓存(Local Metadata Distribution Table，LMDT)的有关内容。首先介绍 LMDT 项的数据结构和内存组织，然后讨论 LMDT 项的状态转换，最后讨论决定 LMDT 管理的关键因素。

3.3.1 元数据分布信息缓存结构

元数据的分布信息缓存以文件索引节点号作为关键字，并通过串表组织起来。除缓存元数据分布信息，降低对后端的压力，缩短请求时延的作用外，元数据分布信息项还需要为文件系统请求并发控制提供支持。从内存结构看，所有分布信息项通过哈希串表组织起来，以方便按照索引节点号的查找。同时，分布信息项按照活跃情况组织在另一个串表上，便于缓存的管理。下面是 LMDT 中一个元数据分布信息项的数据结构：

```
struct lmdt_entry{
    struct list_head le_list;/ * list * /
    struct list_head le_hash;/ * hash list * /

    ino_tle_ino;/ * ino# * /
    u32le_host;/ * host identifier, e. g. MS IP * /

    spinloct_t le_lock;/ * for synchmnizcttion * /
    u32 le_state;/ * state * /
    u32le_refcnt;/ * reference count * /

    u32le_linkcnt;
    u32le_insertcnt;
    u32le_deletecnt;
    u32le_renamecnt;
};
```

字段 le_list 和 le_hash 用来组织内存，通过这两个字段，LMDT 的内存组织如图 3 所示。两个串表头，entry_in_use 和 entry_unused，分别用来组织引用计数 le_refcnt 大于 0 和等于 0 的项。

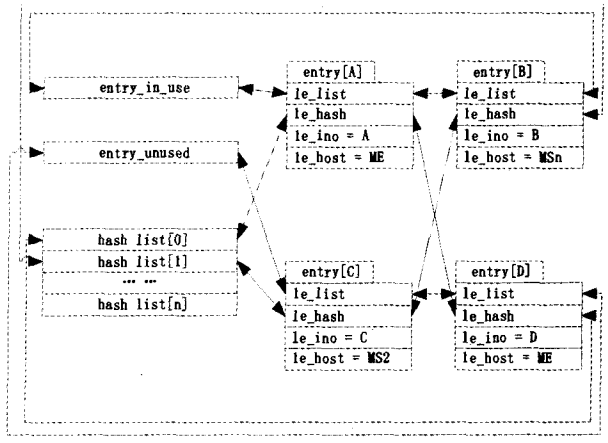

图 3 LMDT 内存结构

le_ino 记录索引节点的 ino#，le_host 记录该索引节点

当前的宿主 MS。le_linkcnt,le_insertcnt,le_deletecnt 和 le_renamecnt 记录本索引节点当前相关的文件系统操作的数量。le_state 记录本分布信息项当前的状态。

3.3.2 元数据分布信息项的状态变化

一个分布信息项可以有 9 个状态:在内存中不存在、向 BS 请求分布过程中、分布在自己并且元数据活跃、没有分布在自己但元数据活跃、分布在自己但元数据不活跃、没有分布在自己且元数据不活跃、主动释放分布信息、被动释放分布信息和释放内存等。如图 4 所示,当元数据请求处理开始时,首先需要根据 ino# 检查相应元数据分布信息项的内容,只有分布在自己的元数据,才能进行相应的元数据请求处理。

当 LMDT 中没有 ino# 对应的分布信息项时,alloc_entry()被用来分配内存被初始化必要信息,但其中的信息不可用,需要调用 map_entry()请求 BS 进行分布决策,此时分布信息项处在 mapping 状态。Mapping 的结果是该项分布在自己或者没有分布在自己,由于项在分配后就处于 le_refcnt > 0 的活跃状态,因此 mapping 完成后项进入 AcOnMe 或者 AcNOnMe 状态。

当请求处理完成不再需要分布信息项时,调用 put_entry()减小项的引用计数。如果引用计数变成 0,则该项变成不活跃,被放入 entry_unused 串表,处于 InacOnme 或者 InacNOnme,成为缓存管理的缓存项释放备选对象。

当 LMDT 中存在相应的项时,get_entry()增加相应的引用计数,并放到 entry_in_use 串表中,分布信息从 InacOnme 或者 InacNOnme 重新回到 AcOnme 或者 AcNOnme 状态。整个分布信息项在 entry_in_use 和 entry_unused 上的转换过程中,它始终通过 le_hash 串在相应的 hash 串表上,通过哈希串表的查找能找到该项。

当 AS 根据访问结果要求某个 MS 强制刷新元数据分布信息项的信息时,revalidate_entry()被触发,MS 向 BS 请求该项的最新信息,同样进入 mapping 状态。

当其他服务器要求本 MS 释放对该项的管理权限时,它通过消息通知本 MS 释放权限。此时该分布信息项通过 release_host_right()进入 releasing 状态。

不活跃的项将被释放内存。对于处在 InacOnme 状态的元数据分布信息项,MS 需要调用 unmap_entry()主动通知 BS 释放该索引节点的分布决策,分布信息项进入 unmapping 状态。Unmap_entry()成功完成后,分布信息项从内存中释放。

当该分布信息项从内存中释放后,该 ino# 对应的分布信息项不复存在,后续访问需要通过 alloc_entry()重新分配新的分布信息项。

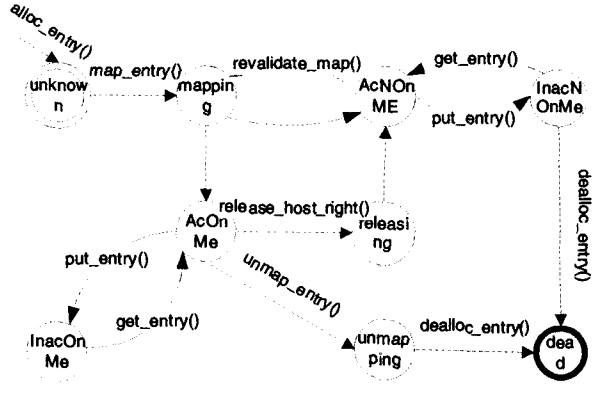

图 4 LMDT 元数据分布信息项状态变迁图

3.3.3 元数据分布信息缓存管理因素

release_host_right()是其他服务器要求 MS 释放对 ino# 的管理权的时机,unmap_entry()是 MS 主动释放对 ino# 的管理权的时机。通过主动释放机制,其他服务器需要该 ino# 管理权限的请求时延将降低,但本 MS 后续访问该 ino# 的元数据的时延将因为缓存中没有相应信息而增加,如何管理元数据分布信息项的释放将影响缓存的命中率,进而影响 BS 的负载和请求处理的时延。

综合多种因素,LMDT 不活跃分布信息项的主动释放由固定时间间隔、缓存所占内存大小和缓存中不活跃分布信息项的数目三个因素决定。在实现上,利用 Linux 内核线程机制完成不活跃项的释放。内核线程的每次工作将释放掉预先设定数量的不活跃项。每隔固定时间间隔,或者缓存总数量超过设定值,或者不活跃数量超过设定值时,内核线程都将被唤醒工作。

根据已有研究结果,由于用户的访问具有极强的局部性,分布在本 MS 的元数据被再次访问的概率要远高于没有分布在本 MS 的元数据,因此,内核线程每次释放首先选择没有分布在自己的不活跃元数据分布信息项,只有在必要的时候才选择分布在自己的不活跃元数据分布信息项。

4 性能评估

不同的缓存管理参数将带来不同的缓存命中率,影响请求处理的时间延迟和 BS 的负载处理。为评估 MS 元数据分布信息缓存对 MS 需要的与 BS 通信数量和请求访问延迟的影响,我们通过调整影响元数据分布信息缓存管理的三个因素获得 0%,50%,90%,95%,98%和 100%的缓存命中率,统计客户请求吞吐率和读写吞吐率以及需要的与 BS 的通信数量。

测试由 postmark[3] 驱动,postmark 参数是 10000 个 4kB ~16kB 的文件,10 个子目录,50000 个事务,块读写为 4kB。postmark 首先需要生成 10000 个测试文件集合,在创建时 MS 需要与 BS 进行通信,完成 MS 元数据分布信息缓存的 warm-up。

为避免服务器数量庞大带来的竞争影响,测试环境由 1 个客户端、1 个元数据服务器、1 个存储设备和 1 个绑定服务器构成。每个服务器的操作系统都是 Redhat(r)8.0,每个服务器的 CPU 是 Intel® xeon3.2GHz, 3GB 内存。SN 是 1 个 120GB 的 SATA 硬盘分区。

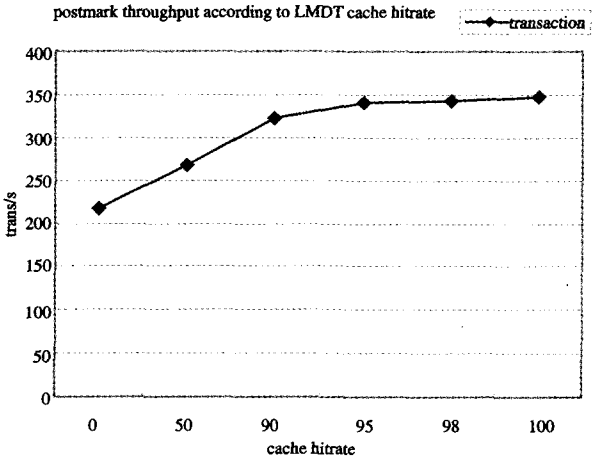

图 5 请求吞吐率随缓存命中率变化图

(下转第 158 页)

169～202
2 Beritino E, Castano S, Ferrai E. Securing XML documents with Author-X. IEEE Internet Computing, May/June 2001. 21～31
3 Bertino E, Sandhu R. Database Security-Concepts, Approaches, and Challenges. IEEE Transactions on Dependable and Secure Computing, 2005, 2(1): 2～19
4 LI Lan, HE Yong-Zhong, FENG Deng-Guo. A fine-grained mandatory access control model for XML documents. Journal of Software, 2004, 15(10): 1528～1537
5 Kuper G, Massacci F, Rassadko N. Generalized XML Security Views. SACMAT 2005. In: 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden, June 2005. 77～84
6 Kudo M, Hada S. XML Document Security based on Provisional Authorization. In: Proceedings of the 7th ACM Conference on Computer and Communication Security, Athens, Greece, 2000. 87～96
7 Liu Mengchi. A logical foundation for XML. In: Proceedings of the 14th Int'l Conf. on Advanced Information Systems Engineering, Toronto, Canada, 2002. 568～583
8 LaPadula L J, Bell D E. Secure computer systems: A mathematical model: [Technical Report 2547 (Volume II)]. The MITRE Corporation, Bedford, Massachussetts, May 1973
9 Bell D E, LaPadula L J. Secure Computer Systems: Mathematical Foundations and Model: [Technical Report M74-244]. The MITRE Corporation, Bedford, Massachussetts, May 1973

(上接第 145 页)

图 5 给出了不同缓存命中率下 postmark 每秒完成的事务数量。从图 5 可以看出，随着缓存命中率的提高，请求吞吐率得到提高。当命中率达到 95%后，缓存不命中带来的请求处理时延对系统的事务吞吐率的影响比较弱，表现出事务吞吐率逐渐平稳的趋势。

同样，我们还获得各个命中率下请求比率，其计算公式是 ratio = MS 向 BS 发起的请求数量 / MS 收到的客户段的请求数量。请求比率趋势图如图 6 所示。每个计算中包含创建测试文件集所需要的 10000 次通信。从图 6 可以看出，缓存命中率将明显影响 BS 的负载压力。

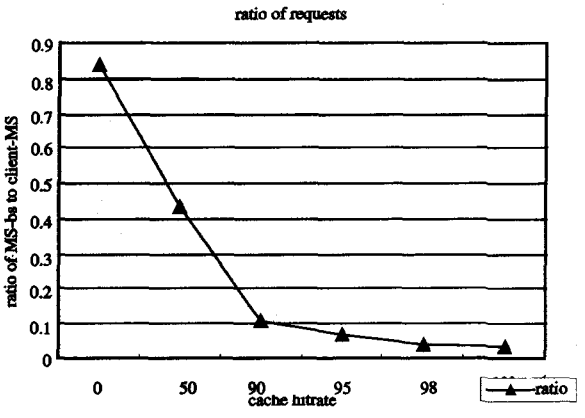

图 6 BS 请求数量随缓存命中率变化图

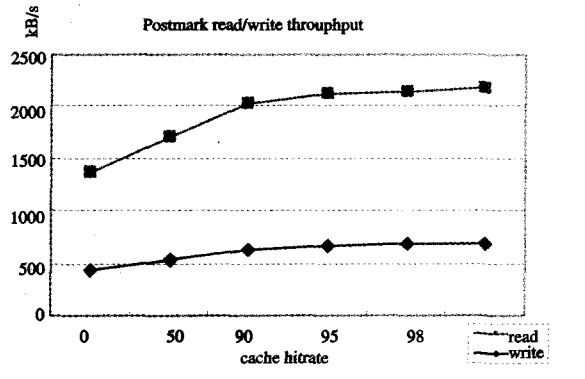

图 7 数据读写吞吐率随缓存命中率变化图

图 7 是同一测试获得的各种缓存命中率下客户端的读写吞吐率。尽管 BWFS 通过文件数据访问的控制流和数据流分离机制给用户提供并发的数据读写，但由于文件数据较小，元数据请求对小文件数据读写吞吐率的影响要比对大文件的数据读写吞吐率的影响明显得多。反应到图 7 上，元数据请求过程的时延对系统读写吞吐率的影响非常明显。同样，当缓存命中率达到 95%后，由于缓存不命中带来的元数据处理时延对整个读写的影响比较弱，系统读写吞吐率表现出平稳的趋势。

5 相关研究

由于网络存储应用的多样化，分布式文件系统元数据服务研究成为热点，其关键问题是如何分布元数据请求负载。已有研究成果可以归纳为两类。一类是文件系统目录子树分区法[4]，目录及其所有子节点的请求由一个元数据服务器完成处理。这种策略人为地限制了用户的请求不能跨服务器完成，并且元数据服务不能根据文件系统目录结构进行深度扩展。第二类是以哈希的方式管理分布[5～7]。哈希法可以加快元数据到服务器的映射过程。但由于哈希法主要以静态因素为函数参数，它不能很好地适应系统动态的规模变化，哈希参数变化的波及面很广，对系统的扩展能力具有很大的限制作用。

结束语 本文描述了 BWMMS 的层次元数据管理机制，动态灵活的元数据分布策略。同时，由于各个 MS 的元数据分布信息缓存管理的有效性对元数据请求的处理延迟和元数据请求的并发同步有着至关重要的作用，我们着重描述各个元数据服务器对分布信息缓存的管理，并通过实验评测缓存管理主要因素对元数据请求处理延迟的影响。

在后面的工作中，我们将对系统进行更详尽的评测，将针对多种典型应用进一步验证 BWMMS 动态灵活的元数据分布管理机制的适用性，将进行元数据分布策略的自学习能力研究。

参 考 文 献

1 Ousterhout J K, Costa H D, Harrison D, et al. A trace-driven analysis of the Unix 4.2 BSD file system. In: Proceeding of the 10th ACM Symposium on Operating Systems Principles (SOSP '85), Dec. 1985. 15～24
2 杨德志，黄华，张建刚，许鲁. 大容量、高性能、高扩展能力的蓝鲸分布式文件系统. 计算机研究与发展，2005，42(6)
3 postmark. http://www.netapp.com/tech_library/3022.html
4 Levy E, Silberschatz A. Distributed file systems: Concepts and examples. ACM Computing Surveys, 1990, 22(4)
5 Corbett P F, Feitelso D G. The Vesta parallel file system. ACM Transactions on Computer Systems, 1996, 14(3): 225～264
6 Brandt S A, Miller E L, Long D D E, et al. Efficient Metadata Management in Large Distributed Storage System. In: Proc. of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003
7 Yan J, Zhou Y L, Xiong H, et al. A Design of Metadata Server Cluster in Large Distributed Object-based Storage. In: Proc. of the 21th IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies, 2004