一种基于NAS的镜像文件系统的研究及其在Linux下的实现*)

谢长生 蔡 斌 曹 强 刘朝斌 黄建忠

(华中科技大学计算机学院外存储系统国家重点实验室 武汉430074)

摘 要 全球数据量快速的增长使得NAS得以广泛应用,但是传统的NAS系统在设备发生故障时数据会丢失,难以满足现代存储对系统的高可用性和高可靠性的要求。本文提出了一种用于实现文件镜像功能的NAS系统框架结构,并且在VFS机制的基础之上,在Linux平台下,设计并实现了一种应用于这种结构的镜像文件系统(NASMFS)。它屏蔽了系统底层中各个NAS设备中文件系统的差别,给用户提供了一个安全、可靠的统一的存储目录空间,透明地完成文件镜像功能,在设备发生故障时能够从镜像的文件中恢复数据,从而在可用性、可靠性等方面弥补了传统NAS系统的不足,满足了应用的需要。

关键词 附网存储,镜像文件系统,存储目录空间,NASMFS,VFS,Linux

The Research of NAS-Based Mirror File System and it's Implementation under Linux

XIE Chang-Sheng CAI Bin CAO Qiang LIU Zhao-Bin HUANG Jian-Zhong

(National Storage System Laboratory, School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074)

Abstract The rapid increase of Global data brings the extensive applications of NAS, but the data may lose when the devices occur failure in the traditional structure of NAS, from this point, traditional structure of NAS can not match the demands of a modern storage system in availability and reliability. This paper puts forward a NAS frame structure, which is applied in files mirror, and based on VFS mechanism, designs and implements a file system under Linux platform, named Mirror File System (NASMFS), which is applied in this NAS frame structure. This file system screens the differences of each NAS device's file system in the low-level of a system, and it provides a secure and reliable uniform storage directory space to clients, accomplishes the mirror function transparently as well, so it can restore the data from the mirror files when the devices occur failure. Concerning the availability and reliability of the traditional structure of NAS, this paper offsets these limitations, and meets the demands of applications.

Keywords NAS, Mirror file system, Storage directory space, NASMFS, VFS, Linux

1 引言

随着数据量指数级的增加,数据存储系统受到越来越多的重视,NAS就是其中的一种以文件为中心的存储结构。按照存储网络工业协会(SNIA)的定义:NAS是可以直接连到网络上向用户提供文件级服务的存储设备[1],客户端通过NFS和CIFS协议来访问网络中的NAS设备。但是当NAS系统中的某个设备发生故障时,该设备上的数据就有可能丢失,整个系统的可靠性和可用性得不到保障。

为了能够恢复在NAS设备发生故障时丢失的数据,设计一种对于用户透明的,能够完成文件镜像功能的文件系统就十分重要,从而在设备发生故障时能够从镜像的文件中恢复数据,提高系统的可用性和可靠性。为此,在本文中,首先对Linux中的VFS抽象虚拟文件系统机制进行了深入的剖析;接着提出了一种用于实现文件镜像功能的NAS系统框架结构;然后在VFS机制和系统框架结构的基础之上,设计了应用于该结构的镜像文件系统—NASMFS并且在Linux平台下实现了该文件系统;最后对NASMFS文件系统进行了性能测试和分析。

2 Linux中VFS文件系统机制

Linux为了能支持多种不同的文件系统,使用了VFS抽象虚拟文件系统层将实际的文件系统同操作系统分离开来,它屏蔽了不同文件系统之间的差异,使所有文件系统对操作系统的其他部分和用户程序来说是透明的。VFS的主要实现思路是在内核中提供一个文件系统框架,包括接口函数指针集、管理不同文件系统的数据结构以及各种缓存机制[2]。从图1中可知,VFS提供上下两个方面的接口:VFS上层接口提供给I/O系统的用户使用(包括用户进程和内核的其他管理模块),它们通过该接口使用I/O系统(文件、设备、网络等),如打开、关闭、读、写文件等操作;VFS的下层接口提供给真实文件系统使用,VFS支持的每个真实的文件系统都要提供对这个接口的实现。VFS本身则利用它的下层接口来实现它的上层接口。下层接口由真实的文件系统实现,VFS并不关心具体实现的细节。为了实现上下层接口之间的转换,VFS必须维护许多的数据结构和接口函数指针集:描述文件系统类型的file_system_type结构、记录已安装文件系统信息的vfsmount结构、记录真实文件系统信息的超级块结构和超级块操作函数接口、记录文件或者目录信息的索引节点结构和索引节点操作函数接口、描述文件组织关系的目录项结构和目录项操作函数接口、描述每个打开文件的文件读写上下文结构和文件操作函数接口以及文件的页面缓冲队列机制和文件的页面缓冲队列操作函数接口。这些VFS抽象层的数据结构中的信息来源于真实的文件系统。在注册、安装、打开的过程中,VFS建立起了上下层接口之间的映射关系,从而可以实

*)本文受国家自然科学基金项目(编号:60173043)资助。谢长生 教授,博士生导师,研究方向为基于网络的存储系统,计算机高速接口与通道,采用新原理的超高密度、超高速存储技术。蔡 斌、曹 强、刘朝斌、黄建忠 博士生,研究方向为操作系统,基于IP的存储区域网,网络存储和流媒体存储。

现准确的操作转接；同时 VFS 抽象虚拟文件系统层对不同的底层文件系统统一处理系统调用；实现路径名称到其 inode 索引节点和 dentry 目录项结构的转换也是 VFS 抽象虚拟文件系统层的主要功能。VFS 抽象层与具体文件系统的联系主要就是通过 super_block、inode、dentry、file 这4条主线联系起来的，每条主线都有其特殊的作用。

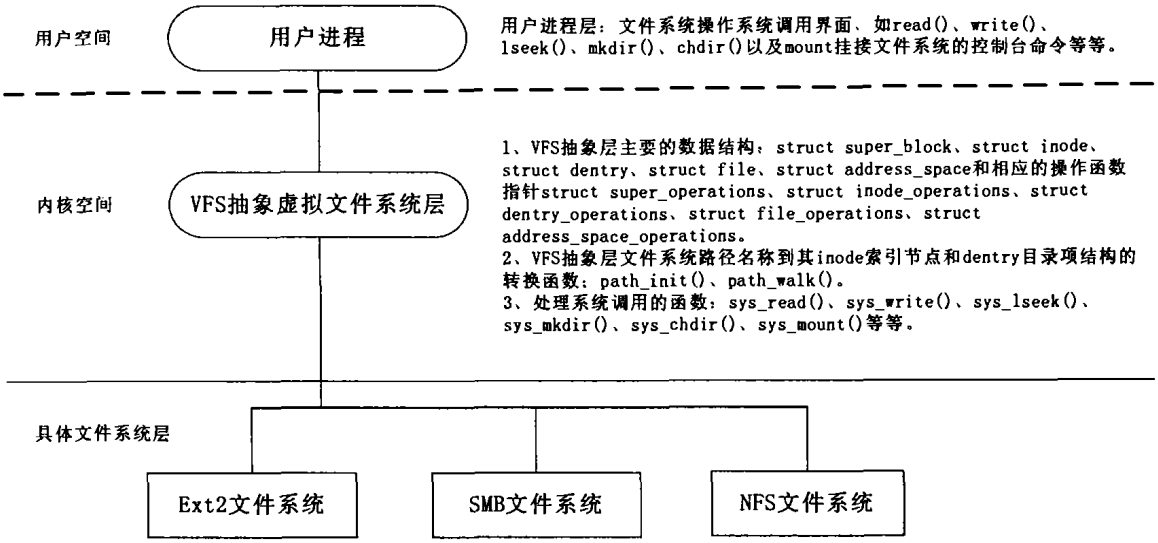

图1　Linux 的 VFS 抽象层和具体文件系统的关系图

3　NASMFS 文件系统的设计及其实现

本节提出了一种用于实现文件镜像功能的 NAS 系统框架结构；然后设计了应用于该结构中的镜像文件系统，即 NASMFS；最后在 Linux 平台下实现了该文件系统。

3.1　系统的整体框架

图2给出了系统的整体结构：客户端首先必须挂接 NAS 镜像文件系统即 NASMFS 文件系统，然后向 NAS 镜像文件服务器提交读写文件的请求；NASMFS 镜像文件系统接受客户端的请求，经过 NASMFS 镜像文件系统层的处理从而向下层各个 NAS 设备发送文件读写的请求，读请求从其中一个 NAS 设备中读取，写请求则将请求写入到每一个 NAS 设备中，文件请求完成后再将结果返回给客户端；每个 NAS 设备均向上层的 NASMFS 镜像文件系统注册其自身的文件系统和可用的目录空间，由这些目录空间组织成 NASMFS 镜像文件系统中统一的存储目录空间，屏蔽了各个 NAS 设备的差别，同时各个 NAS 设备响应 NASMFS 镜像文件系统的读写请求从而实际地完成文件的读写，并且将结果返回给 NAS 镜像文件服务器，再由 NAS 镜像文件服务器将结果返回给提出请求的客户端。

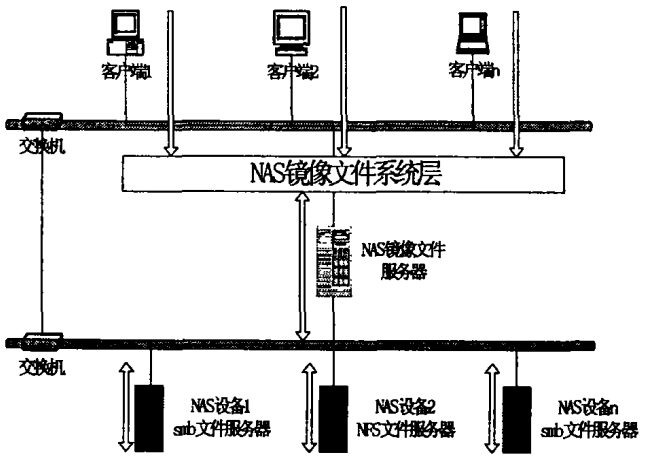

图2　系统整体框架图

在这个系统框架中，NASMFS 镜像文件系统向客户端提供一个统一的存储目录空间[3]，屏蔽了下层的各个 NAS 文件服务器的具体的文件系统的差别，在客户端看来只有 NASMFS 镜像文件系统而并不知道其下的各个 NAS 设备的文件系统。从这种意义上说，NASMFS 镜像文件系统与 VFS 抽象虚拟文件系统层有着共同的地方，从功能上来说像是一个虚拟的文件系统路由器。

3.2　NASMFS 文件系统的设计及其在 Linux 下的实现

VFS 抽象虚拟文件系统有着很好的扩展性[4]，基于 VFS 的机制，在 VFS 抽象文件系统层和具体的 NAS 文件系统（NFS 或者 SMBFS）层之间添加 NASMFS 镜像文件系统层来实现 NAS 系统的文件镜像功能，从而在设备发生故障时能够从镜像的文件中恢复数据，提高系统的可用性和可靠性。

NASMFS 文件系统要提供上下两个层次的接口：上层接口是提供给 VFS 使用——包括向 VFS 注册 NASMFS 文件系统以及向 VFS 安装 NASMFS 文件系统；NASMFS 的下层接口是提供给每个 NAS 设备中真实的文件系统使用（包括下层的每个 NAS 设备向 NASMFS 注册自身的文件系统和可用的目录空间）。为了实现上下层接口之间的转换从而完成 NAS 文件的镜像功能，NASMFS 文件系统本身必须维护许多的数据结构同时还必须实现我们前述的 VFS 标准的接口函数指针集。

NASMFS 文件系统的结构非常清晰，其算法流程如图3所示。文件读写的基本流程是：在用户进程对 NASMFS 文件系统提出读写请求后，VFS 将抽象层的数据结构（*super_block*、*inode*、*dentry*、*file*、*address_space*）与 NASMFS 文件系统的数据结构（*nasmfs_dir_entry*）关联起来；在 NASMFS 文件系统层则在下层的 NAS 设备向它注册的基础之上，通过它自身的操作函数指针集（*nasmfs_dir_inode_operations*、*nasmfs_file_operations*）中的 *lookup*() 函数定位到每个下层的 NAS 设备中的文件系统中的目录空间，然后通过 NAS 设备中的具体的文件系统启动设备的输入/输出操作，实现设备上文件的读、写、更新等操作。操作完成之后，由下层的 NAS 设备的文件系统在适当的时机调用 VFS 的更新例程，将改变的数据从内存中全部写回外部设备。在 NASMFS 文件系统中，文件读写的整个流程由 VFS 提供的内存索引节点缓冲区、内存目录项缓冲区和文件页面缓冲区以及由块设备缓冲区提供的数据块缓冲区在内存中完成对索引节点、目录项和

数据块的操作，减少了读取外设的操作次数，在一定程度上保证了文件系统的性能。

在 Linux 下的实现首先要定义 NASMFS 文件系统的主要的数据结构即：struct nasmfs_dir_entry，该数据结构包括了 NASMFS 文件系统的主要的信息，主要内容如下：

```
struct nasmfs_dir_entry {
  unsigned short low_ino; //NASMFS 文件系统的节点的 inode 号
  unsigned short namelen; //NASMFS 文件系统的节点名称的长度
  const char * name; //NASMFS 文件系统的节点的名称
  mode_t mode; //节点的类型：目录、常规文件
  unsigned long size; //文件的数据部分的当前大小
  uid_t uid; //用户 id 号
  gid_t gid; //组 id 号
  struct inode_operations * nasmfs_iops; //索引节点操作函数指针集
  struct file_operations * nasmfs_fops; //文件操作函数指针集
  struct nasmfs_dir_entry * next, * parent, * subdir; //管理 NASMFS 文件系统的目录或者文件的链表
  atomic_t count;//节点引用计数
  ……
  };
```

通过 mode 用户能够创建目录或者常规文件；通过 uid 和 gid 能够对每个用户的目录和文件进行分开管理，用户只有对自己创建的目录和文件有访问权限而对其他用户的目录和文件是不可访问的，实现了一定的安全性；*nasmfs_iops* 是 VFS 抽象文件系统的 *inode_operations* 数据结构即一组索引节点的操作函数指针，在 NASMFS 文件系统中实现了 *inode_operations* 接口，通过这组接口用户可以在 NASMFS 文件系统中创建目录和文件，删除目录和文件等一系列文件系统的常用功能，同时还实现了 *permission* 即权限许可控制功能，以对用户的权限进行控制；*nasmfs_fops* 也是 VFS 抽象文件系统层的 *file_operations* 文件接口函数指针集，通过这组接口用户可以在 NASMFS 文件系统中对文件进行打开、读和写操作，同时还实现了 *readdir* 即读取目录项功能；NASMFS 文件系统也是按照树状结构来管理的，通过 *next* 指针将同一层次的节点的队列链接起来，*parent* 是指向其父节点的指针，*subdir* 是指向下属节点的目录项队列，因此通过这3 个指针可以在 NASMFS 文件系统树型结构中非常方便地搜索任意的节点；*count* 是节点的引用计数器，只有 *count* 等于0时该节点才允许被删除。

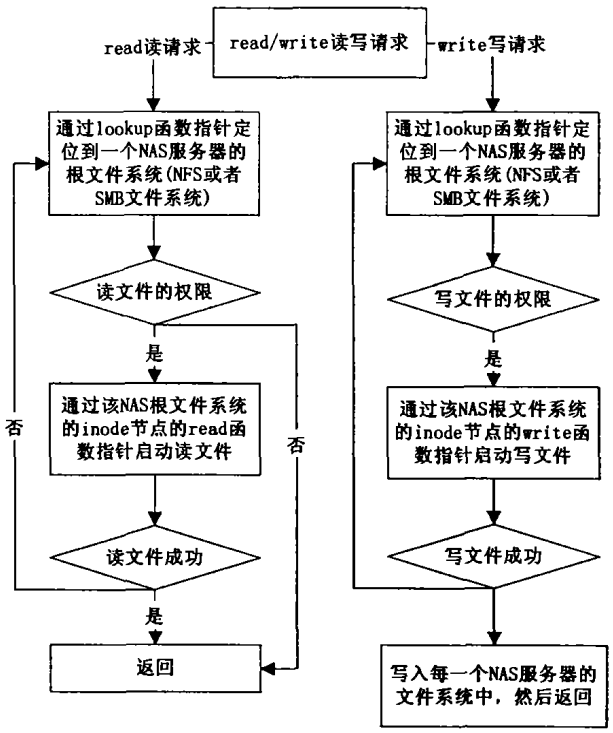

图3 NASMFS 文件系统的算法流程图

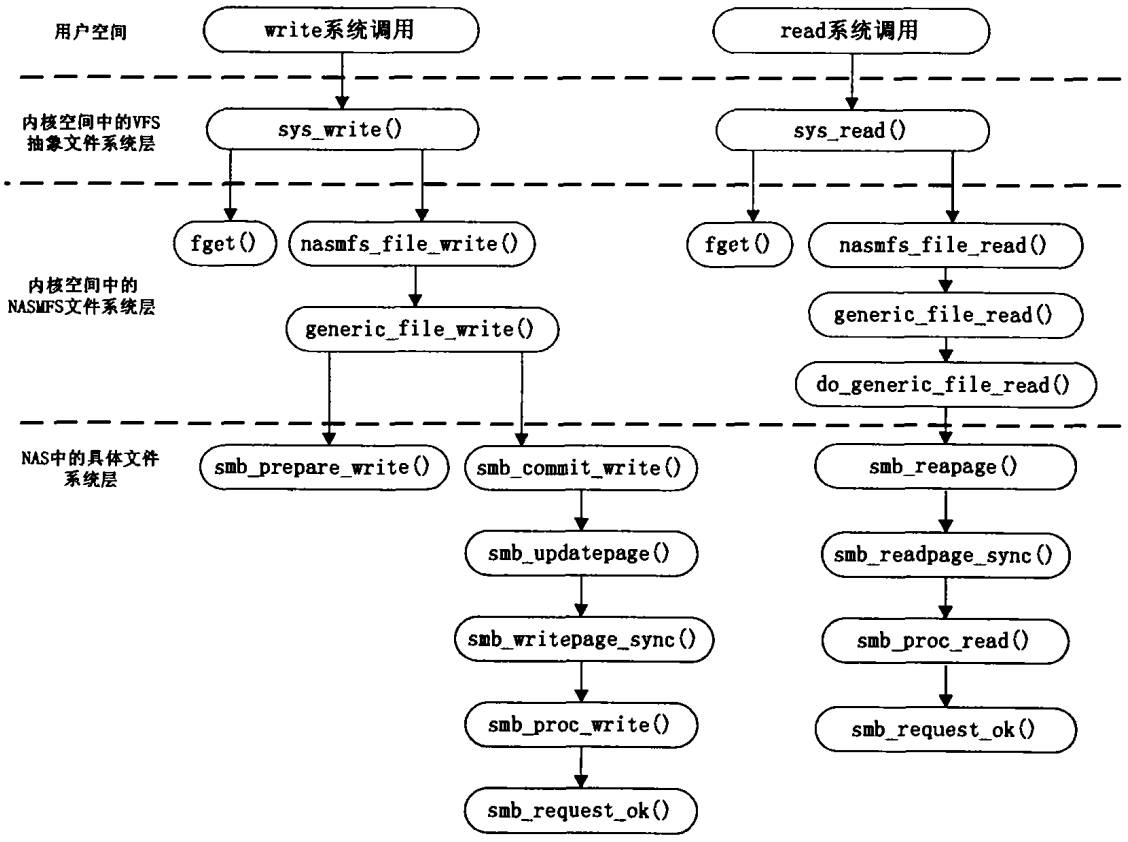

图4 NASMFS 文件系统实现的函数调用关系图

其次，需要定义 *super_operations* 结构的变量 *nasmfs_sops*、*inode_operations* 结构的变量 *nasmfs_dir_inode_operations* 和 *file_operations* 结构的变量 *nasmfs_dir_operations*、*nasmfs_file_operations*。通过实现这4组接口函数指针集才能

够实现 NASMFS 文件系统，从而完成 NAS 文件镜像的功能。在图2中展示了整个系统的结构框架，图4则是对这个系统的函数级的实现，它包含了 NASMFS 文件系统的读写操作，其下层的 NAS 设备的文件系统以 SMB 文件系统为例(NFS 文件系统同)，显示了系统中各个层次的函数级的调用关系。

1)对于读请求：在 VFS 抽象层通过 *file_operations* 数据结构的 *read* 函数指针调用 NASMFS 文件系统的 *read* 函数指针来完成对 NASMFS 文件系统的读操作；几乎所有的文件系统都通过该 *generic_file_read*()函数来进行读文件操作，由 *do_generic_file_read*()进行真正有价值的操作；通过 VFS 抽象层的 *address_space_operations* 数据结构的 *readpage* 函数指针调用 SMB 文件系统的 *readpage*()函数来完成读文件操作。

2)对于写请求：在 VFS 抽象层通过 *file_operations* 数据结构的 *write* 函数指针调用 NASMFS 文件系统的 *write* 函数指针来完成对 NASMFS 文件系统的写操作；在 NASMFS 文件系统层中先通过 *smb_revalidate_inode*()函数检查 NAS 设备上的该文件的 *inode* 是否变化了，如果该 *inode* 改变了，则使本地的 *cache* 无效；接着通过 *smb_open*()检查文件是否已经打开了。然后调用 *generic_file_write*()；在 NAS 设备上通过 VFS 抽象层的 *address_space_operations* 数据结构中的 *prepare_write* 和 *commit_write* 函数指针来调用 SMB 文件系统中的函数从而最后完成写文件操作。

3.3 NASMFS 文件系统的性能测试与分析

测试环境：

	操作系统	CPU	内存	硬盘	网卡
客户端	RedHat Linux 7.2	赛扬1.7G	256M	迈拓6Y060L0(60G/5400rpm)	RealTek8139
NAS 镜像文件服务器	RedHat Linux 7.2	赛扬1.7G	256M	迈拓6Y060L0(60G/5400rpm)	RealTek8139
NAS 设备1使用 SMB 文件系统	Windows2000 professional	赛扬1.7G	256M	迈拓6Y060L0(60G/5400rpm)	RealTek8139
NAS 设备2使用 NFS 文件系统	RedHat Linux 7.2	赛扬1.7G	256M	迈拓6Y060L0(60G/5400rpm)	RealTek8139

网络环境：由一个100Mbps 交换机连接到以太局域网上。

对 NASMFS 文件系统进行了两组测试：第一组的测试对象是3个文件系统(SMB、NFS、NASMFS-NFS-SMB)，其中 SMB 指 NAS 设备1所采用的文件系统，NFS 指 NAS 设备2所采用的文件系统；NASMFS-NFS-SMB 指将 NASMFS 文件系统模块加载在 NASMFS 镜像文件服务器端的内核层，其下层的两个 NAS 设备则分别使用 SMB 和 NFS 文件系统。然后用测试软件 Bonnie++1.03a 版本在客户端使用200大小的文件分别对 NAS 设备1、NAS 设备2和 NASMFS 镜像文件服务器进行了顺序读取、顺序写入和顺序读写(读与写的比例各占50%)这3种不同类型的测试，其测试结果如图5所示；第二组测试主要测试了不同大小的文件对 NASMFS 文件系统顺序写入性能的影响，测试数据分别由1个200M 大小的文件组成和100,000个2K 大小的文件组成，其测试结果如图6所示。

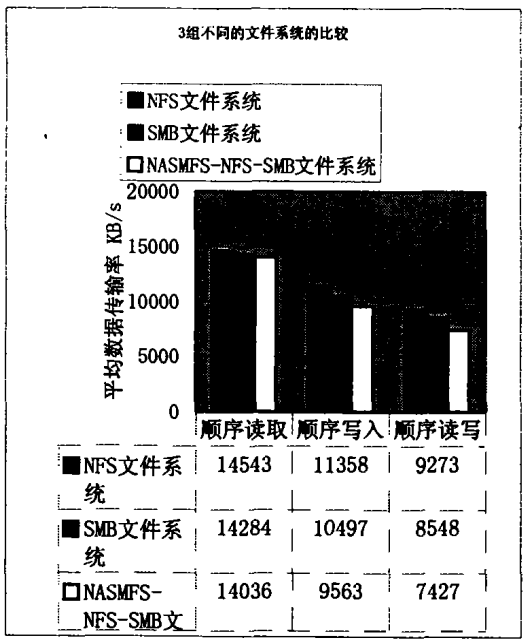

图5 文件系统读写性能测试结果比较图

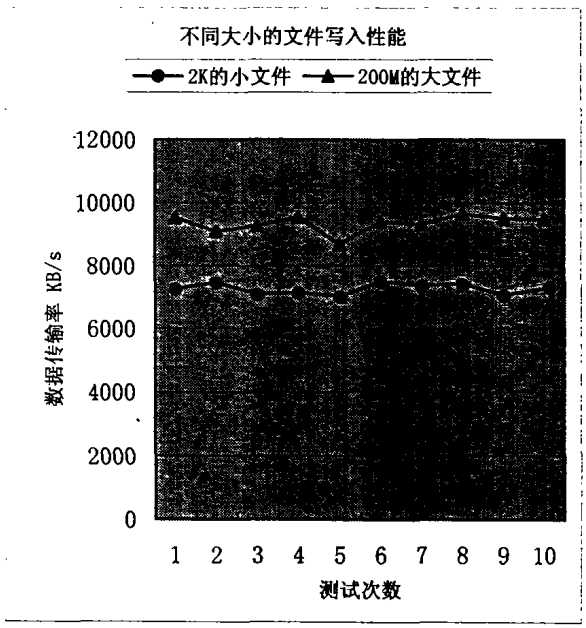

图6 文件大小对性能影响测试结果比较图

从图5中可以看出，对于顺序读取，通过 NAS 镜像文件服务器来读取 NASMFS 文件系统中的文件的性能接近于直接从底层 NAS 设备中读取文件的性能。这是因为对于读取文件操作，NASMFS 文件系统在内核中将请求直接定位到底层设备，性能稍有降低；对于顺序写入，NASMFS 文件系统在内核中将请求依次同步转发到底层中所有的设备中，因此性能比读取操作有了较大的降低。但是从文件镜像的角度来看，在没有 NASMFS 镜像文件系统的情况下，客户端首先向 NFS 写入，然后向 SMB 写入，除去网络延时和客户端的操作时间，完全由用户手工完成一个200M 大小的文件的镜像所需要的时间为200×1024/11358+200×1024/10497=18.031+19.510=37.541s，在使用 NASMFS 文件系统的情况下，时间为200×1024/9563=21.416s，相比之下性能提高了42.953%。显而易见，加上客户端的操作延时和网络开销，采用 NASMFS 文件系统将很大幅度地提高手工镜像的性能，而且这种镜像操作对用户来说是透明的。同时，从图6中可以发现，对小文件的

数据传输率会下降很多。原因是每当收到一个文件必须写入到底层每个NAS设备中，大量的小文件会造成繁重的文件创建工作，因而降低了性能。

结论 全球信息爆炸性的增长使得NAS存储技术得以广泛应用。传统的NAS系统难以满足现代存储高可靠性、高可用性、高性能、动态可扩展性等众多方面的需求。NASMFS镜像文件系统屏蔽了底层中各个NAS设备中文件系统的差别，给用户提供了一个安全、可靠的统一的存储目录空间，透明地完成文件镜像功能，从而在性能、可用性、可靠性等方面弥补了传统NAS系统手工完成文件镜像的不足，满足了应用的要求。

参考文献

1 Gibson G A. network attached storage architecture. COMMUNICATIONS OF THE ACM, 2000, 43(11): 37～45

2 毛德操，胡希明. Linux内核源代码情景分析. 浙江大学出版社，2001

3 Fu Xiangling. Research on Key Technologies of Unified Storage Network: [Ph. D thesis]. Huazhong University of Science and Technology, 2003

4 Bovet D P, Cesati M. Understanding the Linux Kernel, 2nd Edition. O'Reilly, 2002

5 Linux Kernel 2.4.7-10

（上接第56页）

算法参数 N，D 对算法性能有很大影响，我们分别对 $N=5$，$N=10$，$N=15$（当 $D=5$时）和 $D=5$，$D=7$，$D=9$（当 $N=10$时）作出了实验对比，如图2和图3所示。

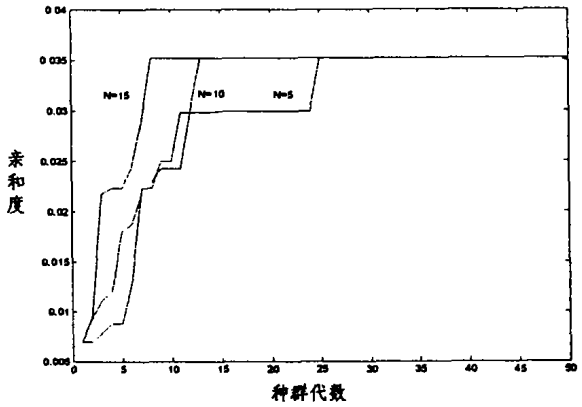

图2 N参数对比图

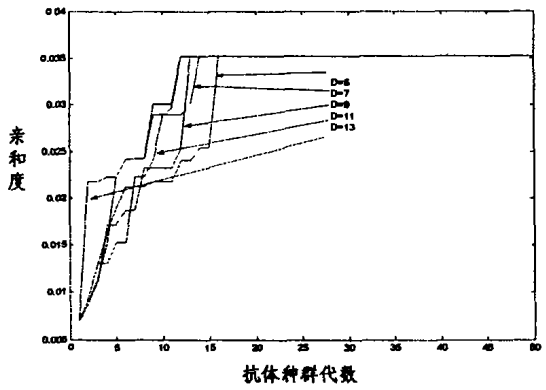

图3 D参数对比图

图2说明，当 N 增加时，算法的收敛速度也在增加。这是因为，如果选取更多的抗体作为生成记忆细胞的种子，那么将增大获得最优解的概率，这符合免疫学原理。虽然下一代抗体的搜索空间得到加大，有助于减小限于局部最优的概率，但在实验中整个算法的运行时间开销也明显增加。因此，N 参数的确定，应根据问题的特定环境。一般来说，如果对时间要求比较紧迫，N 取整个种群大小的10%～15%。

当 D 变大时，引入下一代的随机抗体数量得到增加，这主要是为了增加多样性，它可以有助算法陷入局部最优。但从图3看，参数 D 对算法的收敛速度并没有明显的影响，并且算法的执行时间也没有什么明显的变化。

结论 作为一种智能优化搜索策略，人工免疫系统（AIS）在函数优化、组合优化、调度问题等方面得到了广泛应用并取得了很好的效果。在大多数的情况下，免疫算法取得了比现有启发式算法更好的求解结果[9]。而网格计算是当前网络技术的研究热点，但现在网格技术还没有现成的标准，从网格资源的发现、任务的分配和调度、虚拟组织的管理等等问题都只是提出了一种目标，但具体的实现却没有一种现成的模式，各个开发小组、研究人员都在争相提出自己的独到见解。基于这种情况下，本文将人工免疫系统应用到网格资源中，给出了基于免疫原理的网格资源调度算法。算法从仿真实验来看，是可行和有效的，并且收敛速度较快（一般来说，比遗传算法较快）。

本文的算法是对理想状况下的网格系统的一种调度，但实际网格的资源调度中，还存在很多因素影响调度性能，例如：网络线路流量、资源的使用价格、虚拟组织的策略等等。理想中的网格应该还具有资源预定功能。因此，我们下一步的研究工作就是将上述因素和资源预定功能结合到算法中，以期得到较理想的调度算法。

参考文献

1 Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998

2 Buyya R, Abramson D, Giddy J. Grid Resource Management, Scheduling, and Computational Economy. In: Intl. Workshop on Global and Cluster Computing, Japan, 2000

3 Baker M, Buyya R, Laforenza D. The Grid: International Efforts in Global Computing. In: Intl. Conf. on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, Rome, Italy, 2000

4 Abramson D, Buyya R, Giddy J. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems Journal, Elsevier Science, 2002, 18(8): 1061～1074

5 Buyya R, Abramson D, Giddy J. An Economy Driven Resource Management Architecture for Global Computational Power Grids, In: Intl. Conf. on Paralleland Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, USA, 2000

6 Ibarra O H, Kim C E. Heuristic algorithms for scheduling independent tasks on non-identical processors. Journal of the ACM, 1997, 24(2): 280～289

7 De Castro L N, Von Zuben F J. Clonal selection algorithm with engineering applications. In: GECCO 2000 Workshop proc. 2000. 36～37

8 Morton T E, Pentico D W. Heuristic Scheduling Systems. John Wiley, 1993

9 肖人彬，王磊. 人工免疫系统：原理、模型、分析及展望. 计算机学报，2002，25(12)：1281～1293