VFS层的任务是维护一个表，每个打开的文件在该表中有一个表项。VFS层为每个打开文件保存一个虚拟i节点（或称为v-node）。v节点用来说明文件是本地文件还是远程文件。对于远程文件，v节点提供足够的信息使客户端能够访问它们。对于本地文件，则记录其所在的文件系统和文件的i节点，这是因为现代Linux系统能支持多文件系统（例如ext2fs、/proc、FAT等）。尽管VFS是为了支持NFS而发明的，但多数现代Linux系统将VFS作为操作系统的一个组成部分，不管有没有使用NFS。

为了理解如何使用v节点，我们来跟踪一组顺序执行的mount，open和read调用。要挂载一个远程文件系统，系统管理员（或/etc/rc）调用mount程序，并指明远程目录、远程目录将被挂载到哪个本地目录，以及其他信息。mount程序解析要被挂载的远程目录并找到该目录所在的NFS服务器，然后与该机器连接，请求远程目录的文件句柄。如果该目录存在并可被远程挂载，服务器就返回一个该目录的文件句柄。最后，mount程序调用mount系统调用，将该句柄传递给内核。

然后内核为该远程目录创建一个v节点，并要求客户端代码（图10-36所示）在其内部表中创建一个r节点（remote i-node）来保存该文件句柄。v节点指向r节点。VFS中的每一个v节点最终要么包含一个指向NFS客户端代码中r节点的指针，要么包含指向一个本地文件系统的i节点的指针（在图10-36中用虚线标出）。因此，我们可以从v节点中判断一个文件或目录是本地的还是远程的。如果是本地的，可以定位相应的文件系统和i节点。如果是远程的，可以找到远程主机和文件句柄。

当客户端打开一个远程文件时，在解析路径名的某个时刻，内核会碰到挂载了远程文件系统的目录。内核看到该目录是远程的，并从该目录的v节点中找到指向r节点的指针，然后要求NFS客户端代码打开文件。NFS客户端代码在与该目录关联的远程服务器上查询路径名中剩余的部分，并返回一个文件句柄。它在自己的表中为该远程文件创建一个r节点并报告给VFS层。VFS层在自己的表中为该文件建立一个指向该r节点的v节点。从这里我们再一次看到，每一个打开的文件或目录有一个v节点，要么指向一个r节点，要么指向一个i节点。

返回给调用者的是远程文件的一个文件描述符。VFS层中的表将该文件描述符映射到v节点。注意，服务器端没有创建任何表项。尽管服务器已经准备好在收到请求时提供文件句柄，但它并不记录哪些文件有文件句柄，哪些文件没有。当一个文件句柄发送过来要求访问文件时，它检查该句柄。如果是有效的句柄，就使用它。如果安全策略被启用，验证包含对RPC头中的认证密钥的检验。

当文件描述符被用于后续的系统调用（例如read）时，VFS层先定位相应的v节点，然后根据它确定文件是本地的还是远程的，同时确定哪个i节点或r节点是描述该文件的。然后向服务器发送一个消息，该消息包含句柄、偏移量（由客户端维持，而不是服务器端）和字节数。出于效率方面的考虑，即使要传输的数据很少，客户端和服务器之间的数据传输也使用大数据块，通常是8192字节。

当请求消息到达服务器，它被送到服务器的VFS层，在那里将判断所请求的文件在哪个本地文件系统中。然后，VFS层调用本地文件系统去读取并返回请求的字节。随后，这些数据被传送给客户端。客户端的VFS层接收到它所请求的这个8KB块之后，又自动发出对下一个块的请求，这样当我们需要下一个块时就可以很快地得到。这个特性称为预读（read ahead），它极大地提高了性能。

客户端向服务器写文件的过程是类似的。文件也是以8KB块为单位传输。如果一个write系统调用提供的数据少于8KB，则数据在客户端本地累积，直到达到8KB时才发送给服务器。当然，当文件关闭时，所有的数据都立即发送给服务器。

另一个用来改善性能的技术是缓存，与在通常的UNIX系统中的用法一样。服务器缓存数据以避免磁盘访问，但这对客户端而言是不可见的。客户端维护两个缓存：一个缓存文件属性（i节点），另一个缓存文件数据。当需要i节点或文件块时，就在缓存中检查有无符合的数据。如果有，就可以避免网络流量了。

客户端缓存对性能提升起到很大帮助的同时，也带来了一些令人讨厌的问题。假设两个客户端都缓存了同一个文件块，并且其中一个客户端修改了它。当另一个客户读该块时，它读到的是旧的数据值。这时缓存是不一致的。

考虑到这个问题可能带来的严重性后果，NFS实现做了一些事情来缓解这一问题。第一，为每个缓存了的块关联一个定时器。当定时器到期时，缓存的项目就被丢弃。通常，数据块的时间是3秒，目录块的时间是30秒。这稍微减少了一些风险。另外，当打开一个有缓存的文件时，会向服务器发送一个消