nfs(5) - Linux man page

原文

nfs(5) - Linux man page

挂载参数

所有版本都支持的参数

ac / noac

选择客户端是否可以缓存文件属性。如果没有指定任何选项（或者指定 ac），则客户端缓存文件属性。

为了提高性能，NFS 客户端对文件属性进行缓存。每隔几秒钟，NFS
客户端就会检查服务端上每个文件属性的版本是否有更新。在客户端上次检查后而再次检查之前，这中间的一小段时间，即使服务端文件属性发生了变化，客户端也是无法探测到的。noac 选项阻止客户机缓存文件属性，以便应用程序可以更快地检测到服务端上的文件更改。

除了防止客户端缓存文件属性外，noac 选项还强制应用程序的写操作变得同步，以便对文件的本地更改立即在服务器上可见。这样，其他客户端在检查文件属性时可以快速检测到最近的写入。

使用 noac 选项可以在访问相同文件的 NFS 客户端之间提供更好的缓存一致性，但它会导致显著的性能损失。因此，我们鼓励明智地使用文件锁。数据和元数据一致性章节包含了对这些权衡的详细讨论。

> 关于以下 4 个超时时间，每隔 min 时间，客户端都会去服务端请求最新的数据。而 max 时间相当于强制更新，超过这个时间，强制从服务端更新

acregmin=n

NFS 客户端缓存文件属性的最短时间（单位:秒）。该选项默认值为 3 秒。

acregmin=n

NFS 客户端缓存文件属性的最长时间（单位:秒）。该选项默认值为 60 秒。

acdirmin=n

NFS 客户端缓存文件属性的最短时间（单位:秒）。该选项默认值为 30 秒。

acdirmax=n

NFS 客户端缓存文件属性的最长时间（单位:秒）。该选项默认值为 60 秒。

actimeo=n

使用 actimeo 将 acregmin、acregmax、acdirmin 和 acdirmax 设置为相同的值。如果未指定此选项，NFS 客户端将使用上面列出的每个选项的默认值。

lookupcache=mode

指定内核如何管理给定挂载点的目录项（directory entries）缓存。mode 可以是 all、none、pos 或者 positive。内核 2.6.28 以及更高版本中支持此选项。

Linux NFS 客户端缓存所有 NFS LOOKUP 请求的结果。如果请求的目录项在服务端存在，我们将这一结果称为 positive。如果请求的目录项在服务端不存在，我们将这一结果称为 negative。

如果未指定此选项，或者指定为 all，则客户端假定以上两种类型的目录项缓存都有效，直到它们的父目录项缓存失效。

如果指定了 pos 或 positive，客户端假设 positive 的目录项在其父目录项的缓存属性过期之前都是有效的，但在应用程序可以使用它们之前，总是重新验证 negative 的目录项。

如果指定了 none，客户端将在应用使用这两种类型的目录缓存项之前重新验证它们。这允许快速检测由其他客户端创建或删除的文件，但可能会影响应用程序和服务器性能。

数据和元数据一致性章节包含了对这些权衡的详细讨论。

只有 NFS v2 和 v3 支持的参数

cto / nocto

选择是否使用close-to-oepn 缓存一致性语义。如果没有指定任何选项（或者指定 cto），客户端将使用 close-to-open 缓存一致性语义。如果指定了 nocto 选项，客户端将使用非标准的启发式方法来确定服务端的文件何时发生了更改。

使用 nocto 选项可以提高只读挂载的性能，但仅当服务器上的数据偶尔发生变化时才应该使用。数据和元数据一致性章节将更详细地讨论此选项的行为。

只有 NFS v4 参数

cto / nocto

选择是否对此挂载点上的 NFS 目录 close-to-open 缓存一致性语义。如果既没有指定 cto 也没有指定 nocto，则默认对目录使用 close-to-open 的缓存一致性语义。

此选项不影响文件数据缓存行为。数据和元数据一致性章节将更详细地讨论此选项的行为。

数据和元数据一致性

Close-to-open 缓存一致性

当应用程序打开存储在 NFS 服务端的文件时，NFS 客户端会检查该文件是否仍然存在于服务器上（此时忽略缓存），并通过发送 GETATTR 或 ACCESS 请求检测是否允许打开该文件。当应用程序关闭该文件时，NFS 客户端会将所有更改缓存写回该文件，以便下一个客户端打开时可以查看到这些更改。这也使 NFS 客户端有机会通过 close(2) 的返回代码向应用程序报告任何服务端写入的错误。在打开时检查并在关闭时时刷新的行为称为 close-to-open 缓存一致性。

属性缓存

使用 noac 挂载选项可以在多个客户端之间实现属性缓存一致性。几乎每个文件系统操作都会检查文件属性信息。客户端将此信息缓存一段时间，以减少网络和服务器负载。当 noac 生效时，客户端的文件属性缓存将被禁用，因此每个需要检查文件属性的操作都将被迫请求服务端。这允许客户端以许多额外的网络操作为代价，达到非常快速地看到文件的更改的目的。
注意不要将 noac 选项与 “no data caching” 混淆。noac 挂载选项阻止客户端缓存文件元数据，但是仍然存在竞争，可能导致客户端和服务端之间的数据缓存不一致。

NFS 协议没有设计成在没有某种类型的应用程序序列化的情况下支持真正的集群文件系统缓存一致性。如果客户端之间需要绝对的缓存一致性，应用程序应该使用文件锁。或者，应用程序也可以使用 O_DIRECT 标志打开文件，从而完全禁用数据缓存。

目录项缓存

Linux NFS 客户端缓存所有 NFS LOOKUP 请求的结果。如果服务端存在请求的目录项，则将结果称为 positive。如果请求的目录项在服务端不存在（即服务端返回 ENOENT），则将结果称为 negative。为了检测目录项何时在服务端添加或删除，Linux NFS 客户端会监视目录的 mtime。如果客户端检测到目录的 mtime 发生了变化，客户端将删除该目录下的所有 LOOKUP 结果缓存。由于目录的 mtime 是一个缓存属性，客户端可能需要一段时间才能注意到它已更改。有关目录的 mtime 缓存时间的更多信息，请参考 acdirmin、acdirmax 和 noac 挂载选项的描述。

缓存目录项可以提高不与其他客户端的应用程序共享文件的应用程序的性能。然而，使用关于目录的缓存信息可能会干扰在多个客户端上并发运行的应用程序，因为这些应用程序需要快速检测文件的创建或删除。lookupcache 挂载选项允许对目录条目缓存行为进行一些调优。

在内核发布 2.6.28 之前，Linux NFS 客户端只跟踪 positive 的查找结果。这允许应用程序快速检测其他客户端创建的新目录项，同时仍然提供缓存的一些性能优势。如果应用程序依赖于 Linux NFS 客户端前面的查找缓存行为，则可以使用 lookupcache=positive。

如果客户端忽略它的缓存并与服务器验证每个应用程序查找请求，那么该客户端可以立即检测到其他客户端创建或删除了新的目录项。可以使用 lookupcache=none 来指定这种行为。如果客户端不缓存目录项，则需要额外的 NFS 请求，这会导致性能损失。禁用 lookup 缓存所导致的性能损失应该比使用 noac 要小，并且对 NFS 客户端如何缓存文件属性没有影响。

NFS v4 缓存特性

NFS v4 客户端的数据和元数据缓存行为与早期版本类似。但是，NFS v4 增加了两个改进缓存行为的特性：更改属性（change attributes）和文件委托（file delegation）。

更改属性（change attributes）是 NFS 文件和目录元数据的一个新部分，用于跟踪数据更改。它取代了使用文件的修改和更改时间戳作为客户端验证其缓存内容的一种方式。但是，更改属性（change

attributes）与服务器或客户机上的时间戳解析无关。

文件委托（file delegation）是 NFS v4
客户端和服务端之间的契约，允许客户端临时处理文件，就像没有其他客户端正在访问它一样。如果另一个客户端试图访问该文件，服务器承诺将通知客户端（通过回调请求）。一旦将文件委托给客户端，客户端就可以主动缓存该文件的数据和元数据，而无需与服务器联系。

文件委托（file delegation）有两种形式：读和写。读委托（read delegation）意味着当有其他客户端想要写入该文件，服务端会通知客户端。写委托（write delegation）意味着当有其他客户端读或写入该文件时，服务端会通知客户端。

服务端在打开文件时授予文件委托（file delegation），并且可以在任何时候当另一个客户端想要访问与已经授予的任何委托（delegation）冲突的文件时收回委托（delegation）。NFS v4
不支持对目录的委托（delegation）。

为了支持委托回调，在客户端与服务器初次连接时，服务器会检查到客户端到服务端的网络路径（即客户端起
service，服务端能否正常请求）。如果无法与客户端建立联系，服务器就不向该客户端授予任何委托。

总结

- 在 NFS v4 之前，元数据缓存变更检测属于被动模式，xxxx
- 在 NFS v4 之后，元数据缓存变更检测变成了主动的模式（change attributes），一致性得到了保证，以及性能上得到了提升（delegation），xxx