第二十讲：分布式系统

第 2 节：分布式文件系统

向勇、陈渝、李国良

清华大学计算机系

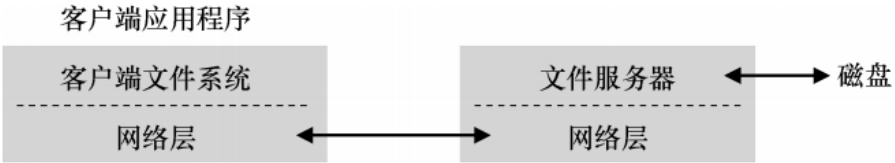
xyong,yuchen,liguoliang@tsinghua.edu.cn

2021 年 5 月 10 日

提纲

分布式文件系统

分布式客户端/服务器计算的首次使用之一，是在分布式文件系统领域。
关键问题：如何构建分布式文件系统
对于客户端应用程序，分布式文件系统似乎与本地文件系统没有任何不同

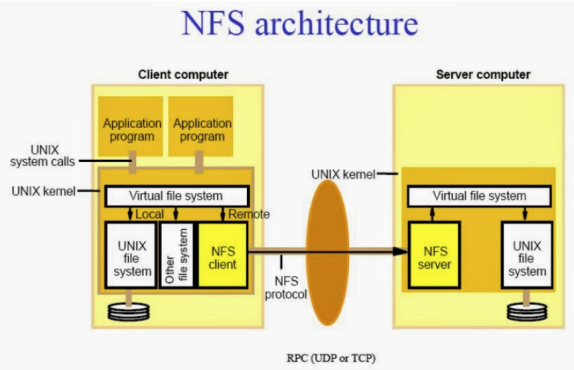


Sun 的 NFS

最早且相当成功的分布式系统之一是由 Sun Microsystems 开发的，被称为 Sun 网络文件系统（或 NFS）

Sun 的 NFS

最早且相当成功的分布式系统之一是由 Sun Microsystems 开发的，被称为 Sun 网络文件系统（或 NFS）在定义 NFS 时,Sun 开发了一种开放协议（open protocol），它只是指定了客户端和服务器用于通信的确切消息格式，而不是构建专有的封闭系统。



NFSv2 协议

关键技术

- 目标：简单快速的服务器崩溃恢复

NFSv2 协议

关键技术

- 目标：简单快速的服务器崩溃恢复
- 快速崩溃恢复的关键：无状态
- 简而言之，服务器不会追踪客户正在做什么

```
char buffer[MAX];
int fd = open("foo", O_RDONLY); // get descriptor "fd"
read(fd, buffer, MAX);          // read MAX bytes from foo (via fd)
read(fd, buffer, MAX);          // read MAX bytes from foo
...
read(fd, buffer, MAX);          // read MAX bytes from foo
close(fd);                      // close file
```

有状态（stateful）协议的示例。文件描述符是客户端和服务器之间的共享状态

NFSv2 协议

问题：共享状态使崩溃恢复变得复杂

- Srv: 在第一次读取完成后，但在客户端发出第二次读取之前，服务器崩溃。
- Srv: 服务器启动并再次运行后，客户端会发出第二次读取，但服务器不知道 fd 指的是哪个文件？
- Client: 一个打开文件然后崩溃的客户端：open() 在服务器上用掉了一个文件描述符，服务器如何关闭给定的文件呢？

```
char buffer[MAX];
int fd = open("foo", O_RDONLY); // get descriptor "fd"
read(fd, buffer, MAX);          // read MAX bytes from foo (via fd)
read(fd, buffer, MAX);          // read MAX bytes from foo
...
read(fd, buffer, MAX);          // read MAX bytes from foo
close(fd);                      // close file
```

NFSv2 协议

挑战：如何定义无状态文件协议，让它既无状态，又支持 POSIX 文件系统 API？

NFSv2 协议

挑战：如何定义无状态文件协议，让它既无状态，又支持 POSIX 文件系统 API？

- 关键是文件句柄（file handle）。文件句柄用于唯一地描述文件或目录。因此，许多协议请求包括一个文件句柄。
- 服务器启动并再次运行后，客户端会发出第二次读取，但服务器不知道 fd 指的是哪个文件？
- 一个打开文件然后崩溃的客户端：open() 在服务器上用掉了一个文件描述符，服务器如何关闭给定的文件呢？

```
char buffer[MAX];
int fd = open("foo", O_RDONLY); // get descriptor "fd"
read(fd, buffer, MAX);          // read MAX bytes from foo (via fd)
read(fd, buffer, MAX);          // read MAX bytes from foo
...
read(fd, buffer, MAX);          // read MAX bytes from foo
close(fd);                      // close file
```

NFSv2 协议

NFSv2 协议 part1

```
NFSPROC_GETATTR
  expects: file handle
  returns: attributes
NFSPROC_SETATTR
  expects: file handle, attributes
  returns: nothing
NFSPROC_LOOKUP
  expects: directory file handle, name of file/directory to look up
  returns: file handle
NFSPROC_READ
  expects: file handle, offset, count
  returns: data, attributes
```

NFSv2 协议

NFSv2 协议 part2

```
NFSPROC_WRITE
  expects: file handle, offset, count, data
  returns: attributes
NFSPROC_CREATE
  expects: directory file handle, name of file, attributes
  returns: nothing
NFSPROC_REMOVE
  expects: directory file handle, name of file to be removed
  returns: nothing
NFSPROC_MKDIR
  expects: directory file handle, name of directory, attributes
  returns: file handle
NFSPROC_RMDIR
  expects: directory file handle, name of directory to be removed
  returns: nothing
```

NFSv2 协议

读取文件：客户端和文件服务器的操作 part1

客户端	服务器
fd = open(“/foo”,⋯); 发送 LOOKUP (rootdir FH, “foo”)	
	接收 LOOKUP 请求 在 root 目录中查找“foo” 返回 foo 的 FH + 属性
接收 LOOKUP 回复 在打开文件表中分配文件描述符在表中保存 foo 的 FH 保存当前文件位置 (0) 向应用程序返回文件描述符	
read(fd, buffer, MAX); 用 fd 检索打开文件列表 取得 NFS 文件句柄(FH) 使用当前文件位置作为偏移量 发送 READ(FH, offset=0, count=MAX)	

NFSv2 协议

读取文件：客户端和文件服务器的操作 part2

	接收 READ 请求 利用 FH 获取卷/ inode 号 从磁盘（或缓存）读取 inode 计算块位置（利用偏移量） 从磁盘（或缓存）读取数据 向客户端返回数据
接收 READ 回复 更新文件位置（+读取的字节数） 设置当前文件位置= MAX 向应用程序返回数据/错误代码	
read(fd, buffer, MAX); 除了偏移量=MAX，设置当前文件位置= 2*MAX 外，都一样	
read(fd, buffer, MAX); 除了偏移量=2*MAX，设置当前文件位置= 3*MAX 外，都一样	
close(fd); 只需要清理本地数据结构 释放打开文件 表中的描述符“fd” （不需要与服务器通信）	

NFSv2 协议

幂等性

- 如果操作执行多次的效果与执行一次的效果相同，该操作就是幂等的 (idempotent)

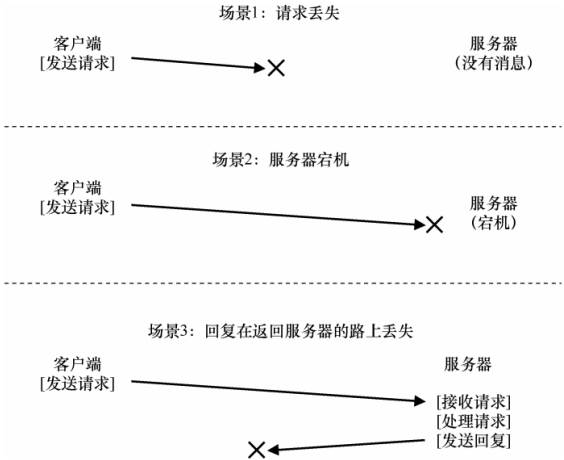
NFSv2 协议

幂等性

- 如果操作执行多次的效果与执行一次的效果相同，该操作就是幂等的 (idempotent)
- 如果将值在内存位置存储 3 次，与存储一次一样。因此“将值存储到内存中”是一种幂等操作。
- 如果将计数器递增 3 次，它的数量就会与递增一次不同。
- LOOKUP 和 READ 请求是简单幂等的，因为它们只从文件服务器读取信息而不更新它。
- WRITE 请求也是幂等的。WRITE 消息包含数据、计数和（重要的）写入数据的确切偏移量。因此，可以重复多次写入，因为多次写入的结果与单次的结果相同。

NFSv2 协议

利用幂等操作处理服务器故障 (3 种类型的丢失)



NFSv2 协议

客户端以统一的幂等操作方式处理了消息丢失和服务器故障

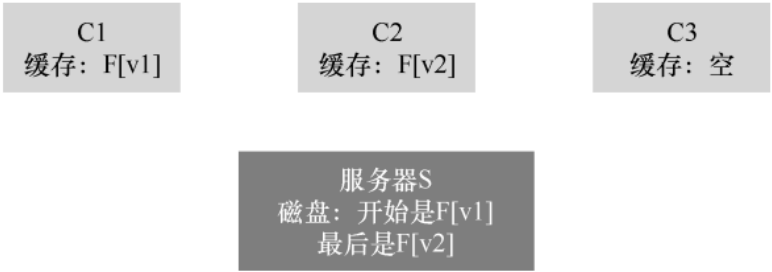
- 如果 WRITE 请求丢失（上面的第一种情况），客户端将重试它，服务器将执行写入，一切都会好。
- 如果在请求发送时，服务器恰好关闭，但在第二个请求发送时，服务器已重启并继续运行，则又会如愿执行（第二种情况）。
- 服务器可能实际上收到了 WRITE 请求，发出写入磁盘并发送回复。此回复可能会丢失（第三种情况），导致客户端重新发送请求。当服务器再次收到请求时，它就会执行相同的操作：将数据写入磁盘，并回复它已完成该操作。如果客户端这次收到了回复，则一切正常

一些操作 (mkdir) 很难成为幂等的

NFSv2 协议

提高性能：客户端缓存

- NFS 客户端文件系统缓存文件数据（和元数据）
- 缓存还可用作写入的临时缓冲区



NFSv2 协议

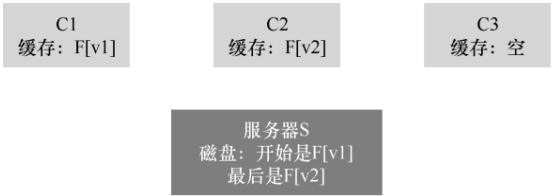
潜在问题：缓存一致性问题

- “更新可见性（update visibility）”问题：来自一个客户端的更新，什么时候被其他客户端看见？

NFSv2 协议

潜在问题：缓存一致性问题

- “更新可见性（update visibility）”问题：来自一个客户端的更新，什么时候被其他客户端看见？
- “陈旧的缓存（stale cache）“问题：C2 最终将它的写入发送给文件服务器，因此服务器具有最新版本（F[v2]）。但是，C1 的缓存中仍然是 F[v1]。如果运行在 C1 上的程序读了文件 F，它将获得过时的版本（F [v1]），而不是最新的版本（F [v2]）

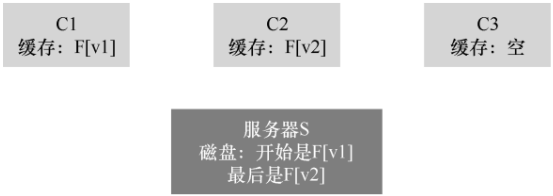


NFSv2 协议

解决方法：缓存一致性问题

“更新可见性（update visibility）”问题：一个客户端的更新何时被其他客户端看见？

- 客户端实现称为“关闭时刷新”（flush-on-close，即 close-to-open）的一致性语义。具体来说，当应用程序写入文件并随后关闭文件时，客户端将所有更新（即缓存中的脏页面）刷新到服务器。通过关闭时刷新的一致性，NFS 可确保后续从另一个节点打开文件，会看到最新的文件版本。



NFSv2 协议

潜在问题：缓存一致性问题
“陈旧的缓存（stale cache）“问题：C2 最终将它的写入发送给文件服务器，因此服务器具有最新版本（F[v2]）。但是，C1 的缓存中仍然是 F[v1]。如果运行在 C1 上的程序读了文件 F，它将获得过时的版本（F [v1]），而不是最新的版本（F [v2]）

NFSv2 协议

潜在问题：缓存一致性问题
“陈旧的缓存（stale cache）“问题：C2 最终将它的写入发送给文件服务器，因此服务器具有最新版本（F[v2]）。但是，C1 的缓存中仍然是 F[v1]。如果运行在 C1 上的程序读了文件 F，它将获得过时的版本（F [v1]），而不是最新的版本（F [v2]）

- NFSv2 客户端会先检查文件是否已更改，然后再使用其缓存内容。具体来说，在打开文件时，客户端文件系统会发出 GETATTR 请求，以获取文件的属性。
- 属性包含有关服务器上次修改文件的信息。如果文件修改的时间晚于文件提取到客户端缓存的时间，则客户端会让文件无效（invalidate），因此将它从客户端缓存中删除，并确保后续的读取将转向服务器，取得该文件的最新版本。

NFSv2 协议

潜在问题：服务器端写缓冲问题

NFSv2 协议

潜在问题：服务器端写缓冲问题
客户端发出以下写入序列

```
write(fd, a_buffer, size); // fill first block with a's
write(fd, b_buffer, size); // fill second block with b's
write(fd, c_buffer, size); // fill third block with c's
```

这些写入覆盖了文件的3个块，先是a，然后是b，最后是c。因此，如果文件最初看起来像这样：

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
```

我们可能期望这些写入之后的最终结果是这样：x、y和z分别用a、b和c覆盖。

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

NFSv2 协议

潜在问题：服务器端写缓冲问题
假设服务器接收到第一个 WRITE 消息，将它发送到磁盘，并向客户端通知成功。现在假设第二次写入只是缓冲在内存中，服务器在强制写入磁盘之前，也向客户端报告成功。但服务器在写入磁盘之前崩溃了。服务器快速重启，并接收第三个写请求，该请求也成功了。
因此，对于客户端，所有请求都成功了，但文件的内容如下：

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy  <---  oops
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

NFSv2 协议

潜在问题：服务器端写缓冲问题
假设服务器接收到第一个 WRITE 消息，将它发送到磁盘，并向客户端通知成功。现在假设第二次写入只是缓冲在内存中，服务器在强制写入磁盘之前，也向客户端报告成功。但服务器在写入磁盘之前崩溃了。服务器快速重启，并接收第三个写请求，该请求也成功了。
因此，对于客户端，所有请求都成功了，但文件的内容如下：

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy  <---  oops
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

解决方法：NFS 服务器在通知客户端成功之前，将每次写入提交到持久存储。这样做可让客户端在写入期间检测到服务器故障，从而重试，直到它最终成功。

NFSv2 协议 – 小结

- NFS 的核心：服务器的故障要能简单快速地恢复。
- 文件操作的幂等性至关重要，因为客户端可以安全地重试失败的操作，不论服务器是否已执行该请求，都可以这样做。
- 将缓存引入多客户端、单服务器的系统，如何会让事情变得复杂。系统必须解决缓存一致性问题，才能合理地运行。