Frangipani 主要与缓存一致性、分布式事务、分布式崩溃恢复以及它们之间的交互有关

Frangipani 是运行在用户机器内核上的一个模块，假设有一群用户在一个工作站里工作，每个人有其自己的机器。工作站里有名为 Petal 的共享磁盘，这是被所有用户共享的，而不是每台机器一个。Petal 可以通过网络将其数据共享给所有的 Frangipani。当 Frangipani 需要读取或写入时，其会发送一个 RPC 请求给 Petal 服务器，然后 Petal 就会返回相应的数据。

注意，这里的环境是基于用户之间都是彼此信任的。

> 问题：如果这些用户读取一些共享文件，然后对其进行修改，那么如何保证一致性？

使用缓存也是一个不错的方法，当查询一次文件后，将该文件的信息缓存在本地，就不用继续发送 RPC 请求来请求文件了。

Frangipani 还支持 “写回” 策略，这意味着如果我想修改某个共享文件，只要其他用户不去查看这个文件，那么我就可以对该文件进行修改，然后将写操作缓存在本地。

假如现在创建一个文件，这可以在本地缓存中完成，对该文件的读写，只需修改本地机器缓存中的内容即可，一般会在过一段时间后再写入 Petal。这对性能有很大的帮助，至少不需要通过 RPC 来请求所需数据。

要在本地缓存对文件进行读写，需要客户端具有相应的操作文件的逻辑，比如支持文件修改、创建等。Petal 是不清楚客户端上的文件以及文件目录信息的，**所有的复杂性都放在了每个 client 端的 Frangipani 中**。这是一种去中心化的方案。

复杂性和大部分 CPU 时间都在 client 端中，这样，当你为系统添加用户时，这未给你增加一定的 CPU 处理能力，但 Petal 也会增加一些存储负载

第一个难题：假设用户 1 创建了一个文件，它可能得先从 Petal 中获取内容，然后创建文件后，其只需要在本地缓存中对其进行修改即可，但是它并不会马上把修改的内容返回给 Petal，那么此时如果用户 2 请求访问该目录下的文件，我们会希望该用户能看到这个新创建的文件，因为情景可能是：旁边的人说我创建了一个文件，你快去看看，但自己查询的时候却没有这个文件。这是一种强一致性，或者是线性一致性。由于这是涉及缓存的内容，其实也是缓存一致性。

第二个难题：由于文件和目录是共享的，如果两个不同的用户在同一时刻对同一个目录进行修改，比如创建两个名字不同的文件，我们可能希望这会有一个合理的结果，即把这两个文件都创建出来，而不是一个操作覆盖了另一个操作，以及如果两个用户创建了相同名字的文件，这里的竞争要怎么处理

此外就是由于采用的是写回策略，如果用户在写入缓存时，还未写到 Petal 时崩溃了，这可能会影响一些文件，我们不希望其他用户会看到一些不合理的东西，即他们查看该目录时，还能看到那些文件，只是没有自己的修改而已，因此我们需要崩溃恢复的能力。

缓存一致性

Frangipani 的缓存一致性是通过锁来实现的。除了用户主机和 Petal 存储服务器，还有 lock 服务器，其上面有一张表，维护的是文件与其获取者的关系，比如 X 请求操作文件 A，则其需要先去获取文件 A 的锁。这个锁具体上是读写锁，目前我们先把它当作独占锁。

在每个用户主机上，其 Frangipani 模块上也会维护一个 lock 表，其维护的是文件名、锁类型以及所缓存的文件内容的信息。如果一个用户想要去访问某个文件，其需要先去看看自己的 lock 表中有没有相应的信息，若有，则直接去缓存访问该文件，若没有，其需要去 lock 服务器获取该文件的锁，获取锁后，其再去访问 Petal 服务器以获取想要读取的数据。当用户对文件进行操作时，其 lock 表的锁类型会被标记为 busy，当用户结束完操作文件的系统调用的时候，它就会把这个锁标记为 idle（闲置）。但是在 lock 服务器看来，这个用户还是持有这个文件的锁。

只有在工作站持有该数据所对应的锁时，我们才允许它去对数据进行缓存。如果用户释放锁，其会将数据写回到 Petal，然后才会把锁还给 lock 服务器，并将这些条目从该用户的 lock 表中删除。

我们不允许两个人持有一样的锁（这里指的是独占锁的前提下）。如果用户想要创建一个新的文件 Y，并对其进行相应的修改操作，那么其也要先向 lock 服务器获取锁。假设当前这个锁被其他用户占有，那么 lock 服务器会向该用户发送 revoke 消息，告诉该用户有别的用户想要获取锁，如果此时该锁的状态是 idle，则其会将修改后的数据写入 Petal，然后对 lock 服务器进行响应，并说我们可以释放这个锁。如果此时锁的状态是 busy，则其需要等待操作结束才会把锁还给 lock 服务器。lock 服务器收到释放锁的消息后，就会把自己 lock 表中关于该条目的信息删除，并且把锁交给当前要请求的用户。因此后续用户可以看到新修改的值

具体实现上，关于锁我们可以实现为读写锁。即只进行读取文件操作的话，他们可以去共享读锁，如果有用户想要进行写操作，那么得等 lock 服务器回收所有的读锁后才能进行写入。

如果写入缓存后，用户机器崩溃了，丢失的数据就没有了，因此，一般来说每 30 秒左右，用户就会把它们缓存修改过的数据发送给 Petal，这样就算机器崩溃，丢失的顶多也是 30 秒内修改的数据。

事务

Frangipani 如何做到原子性？即我们对文件进行一系列操作的时候，别的用户可能只能看见该文件存不存在，而不能看见其中的一些中间操作步骤。

Frangipani 有事务的概念。即直到用户修改完成后，其他用户才能看到这个修改，在它执行操作的期间，首先其要获取所读取或写入数据所需要的全部锁，直到完成操作后，才会释放这些锁。

所以，Frangipani 利用锁保证了缓存一致性以及实现了事务。

崩溃恢复

假设一个用户持有锁，并且在执行更新操作时发生了崩溃。由于 lock 服务器要求它去释放锁，所以它可能已经将部分修改写入了 Petal，但并不是全部的写操作。

由于崩溃时可能没有写入所有的写操作，所以直接释放锁是不 ok 的，因为它会对任何要来读取这段数据的 reader 隐藏这部分更新。但如果一直不释放锁，等待这些数据的用户就得一直等下去，也不行。

Frangipani 利用 wal 来实现崩溃后的恢复。具体操作为：在用户要把更新的数据发给 Petal 之前，它要先在 Petal 中关于它自己本身的日志追加条目，这些条目表示它执行的这一系列操作，只有这些操作完整落地后，才能把数据发给 Petal。

这里有两点不同：

1. 在大多数事务系统中，只有一个日志，即所有的操作都是放在这个日志中的。而 Frangipani 是每个用户一个日志。
2. 所有的日志不存在用户的机器上，而是统一存在 Petal 上

这样如果用户崩溃了，那么其它用户就可以读到它的日志

Petal 是使用一些 block 来存放每个用户的日志，每个用户以环形队列的方式来使用 Petal 上其所对应的日志空间。它会从日志起点开始写入，当空间用完时可以复用，但是要确保被复用空间上的日志已经不再需要了。

每条日志可能会有单调递增的日志序列号，这是因为如果用户崩溃了，Frangipani 会去扫描其对应的日志，那么可以确保最大的日志序列号就是结尾。还有的信息是对应的 Petal 上的 block 号，日志内还有一个操作数组，用于记录该用户的修改，还有一个版本号（后面会提到）。注意日志并不包含写入到文件内容的数据，只包含用于恢复文件系统的足够的信息。

但一开始日志都是存放在用户自己的机器上的，只有不得不将日志写到 Petal 中时，它才会将日志写回 Petal。因为如果一下把日志全写入，可能需要花太多时间。

> 问题：收到 revoke 消息时需要写完整的 log 吗?
>
> Frangipani 是会把完整的 log 写进去，如果是对于文件 Z 的，其需要发送完整的 Log，但是只需要发送关于 Z 的数据即可。
>
> 不过如果对于一个 Log，该 Log 之后的操作都未涉及 Z，那么我们可以只发送到这里的 Log

在用户收到 revoke 消息时，用户需要将部分日志写到 Petal 中，然后将数据发给 Petal，最后和 lock 服务器通信，释放锁。如果在写入日志前崩溃了，则认为这些修改没有做，那就直接释放锁。如果在写入日志后，发送数据期间崩溃了，lock 服务器向用户发送 revoke 请求，但是没有得到回应，超时后（Frangipani 使用了租约的机制），lock 服务器就会认为，该机器已经崩溃，此时 lock 服务器会告诉别的用户，说该用户已经崩溃，你去读取它的日志，并重放该日志的操作，以确保操作是完整的，用户处理完后，就会通知 lock 服务器，此时 lock 服务器就会释放这些锁。

注意一条日志里的操作只是一部分完整的操作，可能并不是用户执行的全部操作，只有日志中的操作是完整的，才会去执行，不完整就不会去重放。因为我们需要确保 Petal 只能看到完整的操作，而不是看到一个做了一半的操作。

但也有可能是只发送了一部分数据只有就崩溃了，此时我们也不清楚具体执行了哪些操作，只会重新进行这些操作。

下面来看一个场景，用户 1 创建文件 F，最后将其删除了，在删除完这个文件后，用户 2 也创建了一个相同名字的文件 F，在创建完后，用户 1 就崩溃了，此时假设用户 3 的机器负责重放用户 1 对应的日志，那么此时可能就会把用户 2 创建的文件给删除了，这是不合理的，因此我们需要进行一些处理，即不能只是单纯地重放用户的日志。

Frangipani 是通过增加版本号这个信息来解决这个问题的。对于 Petal 存储的数据，它们都会有一个版本号，当用户需要去修改数据时，其也会去读取相应的版本号，在修改完毕发送日志给 Petal 时，其会将之前取到的版本号 + 1，添加到日志条目中一起发送给 Petal，发送修改数据时也会这样做。

那么进行重放操作时，其会先去查看对应的版本号，如果 Petal 对应数据的版本号大于等于日志条目中的版本号，而恢复软件就不会执行该更新操作。

但一种问题可能是，在恢复软件执行恢复操作时，其他用户可能仍然持有该文件的锁。一种不可行的解决方法就是，恢复软件去获取该文件对应的锁，但是如果是发生系统级别的供电故障，比如整个数据中心停电，lock 服务器关于 lock 表的信息就会全部丢失了。

但是其实在进行恢复操作时，不关心锁的情况直接进行读写。原因是：在进行恢复操作时，原来的用户要么释放了锁，要么没释放锁。若释放了锁，那么此时对应数据的版本号就会大于等于日志的版本号，此时恢复软件会忽略这个操作；若没释放锁，则其他用户还不能获取这个锁，所以直接修改也没有什么问题。