容错本身是为了提供高可用性（比如出现故障时还能继续提供服务）。我们所用到的一个工具就是复制。但复制也不是万能的，我们需要知道复制能处理什么样的故障。

复制能处理的故障一般是单台计算机的 fail-stop 故障。fail-stop 指的是出些某些故障时计算机会停止运行，如果服务器断网了，也算一种 fail-stop，因为这样服务器跟坏了没什么区别。复制也能处理一些硬件问题，比如服务器 CPU 过热等，这样可以进行切换。

但复制解决不了某些软件中的 bug 和硬件设计的缺陷，比如 MapReduce 中 Master 节点程序有 bug，那么无论将其复制到多少台机器上一样会有 bug。能解决的问题有比如服务器上运行了一些无关的软件，并且其导致了设备崩溃，这时就可以通过复制解决。

对于复制还有一些限制，以 Primary 和 Backup 为例，我们总是假设其错误是独立的，但如果其错误是有关联的，那么复制就没有任何作用。比如这些机器是相同批次的，那么出错时很可能两台机器会一起出错，或者机器放到一起时，突然发生了断电或者地震等情况。

复制是否值不值得这是一个经济的问题，因为我们需要额外的设备，需要具体情况具体分析

## 状态转移和复制状态机

假设一个服务器有两个副本，我们需要让它们保持同步，一旦 Primary 出现故障，Backup 就可以接管服务。

状态转移的思想是：Primary 会将自己完整的状态拷贝发送给 Backup，VMware FT 没有采用这种复制的方法。若采用了，每过一会，Primary 就会对自身的内存做一大份拷贝，然后通过网络发送给 Backup，为了提升效率，你可以每次只发送上次同步后变更了的内存。

复制状态机基于如下事实：在没有外部事件的干预时，计算机只是一个接一个地执行指令。只有外部事件才会发生一些预期外的事。因此复制状态机是将外部输入从 Primary 发送给 Backup，并让它们从相同的状态，以相同的顺序，在相同的时间看到相同的输入，那么它们会一直互为副本，并且保持一致。

所以状态转移传输的是内存的内容，复制状态机是将来自客户端的操作或者其他外部事件从 Primary 传输到 Backup。复制状态机使用的场景比较多，因为传输请求操作比较小，而传输状态需要传输的东西较多。复制状态机的缺点是其要复杂一些，需要做更多假设，状态转移就简单粗暴些，直接将整个状态复制给你。

VMware FT 论文讨论的都是复制状态机，并且只涉及单核 CPU，目前还不清楚论文中的方案如何扩展到多核处理器中，因为多核机器，两个核交互处理指令的行为是不确定的，因此在多核的机器中，Primary 和 Backup 不一定产生相同的结果。

VMware 在之后推出了一个新的可能完全不同的复制系统，并且可以在多核上工作。这个新系统从我看来使用了状态转移，而不是复制状态机。因为面对多核和并行计算，状态转移更加健壮。如果你使用了一台机器，并且将其内存发送过来了，那么那个内存镜像就是机器的状态，并且不受并行计算的影响，但是复制状态机确实会受并行计算的影响。但是另一方面，我认为这种新的多核方案代价会更高一些。

如果我们构建一个复制状态机的方案，我们需要考虑在什么级别上复制状态，我们对状态的定义是什么。并且如果 Primary 发生故障，我们需要一些切换的方案，并且客户端需要知道应该与别的服务器进行通信。但是几乎不可能设计一个不出现异常现象的切换系统。在理想的环境中，如果Primary故障了，系统会切换到Backup，同时没有一个客户端会注意到这里的切换。这在实际上基本不可能实现。所以，在切换过程中，必然会有异常，我们必须找到一种应对它们的方法。

如果我们创建一个新的副本，我们只能使用状态转移，因为复制状态机只是以一种成本更低的方式来保持同步，但是其本身得先具有相同的状态

GFS 实现的是应用程序级别的复制，即将数据抽象成 Chunk 和 Chunk ID，而 VMware FT 是机器级别的复制，即两个机器的最底层也是一样的。

## VMware FT 工作原理

在硬件上运行一个虚拟机监控器（VMM，Virtual Machine Monitor），可以在硬件上模拟出多个虚拟的计算机，并允许一系列不同的操作系统，其中每一个都有自己的操作系统内核和应用程序

我们在两个服务器运行 VMM，Primary 在其中一个物理服务器上，Backup 在另一个物理服务器上。两台物理服务器上的 VMM 会为每个虚拟机分配一段内存，这两段内存的镜像需要完全一致，这两个服务器连接同一个网络，同时网络中还有一些客户端，多副本服务没有使用本地盘，而是使用了一些 Disk Server（远程盘）。

现在，某个客户端向 Primary 发送了一个请求，这个请求以网络数据包的形式发出。这个网络数据包会产生一个中断，然后中断到达 VMM。VMM 发现这是个发给我们多副本服务的一个请求，然后其会模拟网络数据包的中断，以将相应的数据送给应用程序的 Primary 副本。然后，VMM 会将网络数据包复制一份，并通过网络发送给 Backup 所在的 VMM。

然后 Backup 收到该网络数据包，其也会模拟网络数据包到达的中断，以将数据发送给应用程序的 Backup。所以 Primary 和 Backup 都有了这个数据包，它们有了相同的输入，并且会以相同的方式处理这个输入，并保持同步。

然后在 Primary 中的服务会生成一个回复报文，并通过 VMM 在虚机模拟的虚拟网卡发出。之后 VMM 可以看到这个报文，并将这个报文发送给客户端。此外，Backup 运行了相同顺序的指令，其也会生成回复报文，但这里 Backup 会直接将其丢弃。

VMware FT 论文中将 Primary 到 Backup 之间同步的数据流的通道称之为 Log Channel，从 Primary 发往 Backup 的事件被称为 Log Channel 上的 Log Event/Entry。

当 Primary 因为故障停止运行时，Backup 就不会收到来自 Log Channel 上的 Log 条目。实际上 Backup 每秒可以收到很多的 Log。所以 Primary 虚机还在运行时，Backup 必然可以期望从 Channel 中收到很多消息。所以当其收不到 Log 条目时，Backup 虚机会接管，并且不会再等待来自 Primary 的事件。Backup 的 VMM 会让 Backup 自由执行。Backup 的 VMM 会在网络中做一些处理（可能是发 GARP），让后续的客户端请求发往 Backup 虚机，此时 Backup 也不会丢弃回复报文，而是将其返回给客户端。

## 非确定性事件

如果说有非确定性的事件时，我们怎么让 Primary 和 Backup 同步呢?

非确定性的事件有：

- 客户端输入，我们不知道其什么时候到达，有什么样的内容。我们讨论的系统通过网络进行交互，所以这里的请求都是网络数据包。网络数据包对我们来说有两部分，一个是数据包中的数据，一个是提示数据包送达了的中断，然后操作系统会在处理指令的过程中消费这个中断，这里中断的位置得是一样的，否则可能会出现不一致。
- 比如请求获取当前机器的时间、随机数、获取计算机的 ID。
- 多 CPU 的并发，VMware FT 没有讨论这种情况，并且也不适合这节课的讨论。

所有的事件都要通过 Log Channel，从 Primary 同步到 Backup，论文中没有讨论日志的格式，但可以猜测可能有如下三种信息：

1. 事件发生时的指令序号，我们需要让 Primary 和 Backup 在相同的指令位置看到数据。
2. 日志条目的类型，是普通的网络数据输入或者是随机事件。
3. 数据，如果是网络数据包，则数据就是数据包的内容。如果是随机指令，则数据就是这些指令在 Primary 执行的结果，这样 Backup 就可以伪造指令，并提供与 Primary 相同的结果。

Primary 和 Backup 虚拟机的操作系统需要有一个定时器，需要出发中断来计时。因此这里的计时器必须在 Primary 和 Backup 虚拟机完全相同的位置产生中断，否则这两个虚拟机不会以相同的顺序执行指令。

所以在运行 Primary 虚拟机的服务器上，有一个定时器，这个定时器会计时，生成定时器中断并发送给 VMM。在适当的时候，VMM 会停止 Primary 虚机的指令执行，并记下当前的指令序号，**然后在指令序号的位置插入伪造的模拟定时器中断**，并恢复 Primary 虚拟机的运行。之后，VMM 将指令序号和定时器中断再发送给 Backup 虚拟机。虽然 Backup 虚拟机的 VMM 也可以从自己的定时器接收中断，但是其并没有将中断传给 Backup 虚拟机的操作系统，而是直接忽略。当来自 Primary 的 Log 条目到达时，Backup 虚机的 VMM 配合特殊的 CPU，使得物理服务器在相同的指令序号处产生一个定时器中断，之后 VMM 获取到该中断，并伪造一个假的定时器中断，并将其送入 Backup 虚拟机的操作系统，并且这个定时器中断会出现在与 Primary 相同的指令序号位置。

VMware FT 会维护一个来自 Primary 的 Log 条目的缓冲区，如果缓冲区为空，那么 Backup 是不允许执行指令的，只有缓冲区有数据时才会执行，并且只会执行到 Log 条目对于的指令序号，所以 Backup 不会领先 Primary。

当网络数据包到达时，如果我们没有运行虚拟机，网卡会将网络数据包通过 DMA（Direct Memory Access）的方式送到计算机的关联内存中。由于这个网络数据包是发送给虚拟机的，在虚拟机内的操作系统可能会监听 DMA 并将数据拷贝到虚拟机的内存中（有些操作系统会这样）。

我们不能允许这种情况出现，否则我们会失去对 Primary 虚拟机的时序控制，因为我们也不知道什么时候 Primary 会收到网络数据包。所以实际上，物理服务器的网卡会将网络数据包拷贝到 VMM 的内存，之后网卡中断会发送给 VMM，并说一个网络数据包到达了，此时 VMM 会暂停 Primary 虚拟机，记住当前的指令序号，将整个网络数据包拷贝给 Primary 虚拟机的内存，并模拟一个网卡中断给 Primary。同时将网络数据包和指令序号发送给 Backup，Backup 虚拟机的 VMM 也会在对应的指令序号暂停 Backup 虚拟机，然后将网络数据包拷贝给 Backup 虚拟机，之后在相同的指令序号位置模拟一个网卡中断发送给 Backup 虚拟机。这就是论文中介绍的 Bounce Buffer 机制。

## 输出控制

Primary 回复的真实情况会复杂一些，假设当前客户端对 Primary 有一个请求，将 x 自增1（初始值为10），然后 Primary 执行这个请求，回复客户端说 x 的值为11，并将这个请求发送给 Backup 虚拟机，Backup 执行完后将回复报文丢掉。这是我们期望的。

但如果出现了故障怎么办？在这么课中，我们需要始终考虑故障的最坏场景是什么，故障会导致什么结果？假设在 Primary 回复客户端后崩溃了，并且网络变得不可靠，发送的请求丢包了，Backup 没有看到该请求，所以保存的数据还是 10.

由于 Backup 觉察到 Primary 崩溃了，Backup 接管服务，此时客户端再提出将 x 自增，此时 Backup 将 x 自增，x 值变为 11，并回复客户端。这样客户端就会收到有问题的回答（即两次自增的结果都为11）。

VMware FT 的解决方法是控制输出。直到 Backup 虚拟机确认收到相应的 Log 条目之前，Primary 不允许生成任何输出。所以真正的流程为：

1. 客户端输入到达 Primary。
2. Primary 的 VMM 将输入的拷贝发送给 Backup 虚机的 VMM。该请求在Primary 虚机生成输出之前，就通过网络发往了 Backup，但是过程中有可能丢失。
3. Primary 的 VMM 将输入发送给 Primary 虚拟机，Primary 虚拟机生成了输出。现在 Primary 里的数据已经变成了11，生成的输出也包含了11。但是VMM 此时不会转发这个输出给客户端。
4. **Primary 的 VMM 会等到之前的 Log 被 Backup 虚拟机确认收到了才会将输出转发给客户端。** Backup 的VMM 不用等到 Backup 虚拟机实际执行这个输入，就会发送一个 ACK 给 Primary 的 VMM，Primary 的 VMM 收到这个 ACK 后，才会转发这个输出给客户端。

这里的核心思想是，确保在客户端看到对于请求的响应时，Backup 虚拟机一定也看到了对应的请求，或者说至少在 Backup 的 VMM 中缓存了这个请求。 所以如果丢包的话，Primary 的 VMM 就不会收到这个 ACK，也就不会将输出返回给客户端。

> 如果 Backup 返回的 ACK 丢包了怎么办?
>
> 可能 Primary 会继续发送这个请求，这个请求应该会包括相应的序号，这样 Backup 收到后不至于重复执行该请求，只返回 ACK 即可。

Primary 的等待是非常影响性能的，这几乎是所有复制方案中性能的瓶颈，几乎每一个复制系统都会有这个问题。

所以如果条件允许，人们会更喜欢在更高层级做复制的系统（4.2 最后两段），这样 Primary 虚拟机不至于在每个网络数据包都暂停同步一下，甚至可以对一些只读操作不做暂停。

> 能不能输入送到 Primary，输出从 Backup 送出?
>
> 可以考虑一下这个 idea。

## 重复输出

如果回复报文已经从 VMM 发送给客户端了，然后此时 Primary 虚拟机崩溃了，而 Backup 还未执行到相应的请求，然后 Backup 接管服务，其需要消费 Log 缓冲区中的 Log，以保持与 Primary 相同的状态，这样当 Backup 执行完该请求后，也会给客户端返回一个回复报文（此时不会丢弃该报文，因为已经接管了）。这样客户端就会收到两个回复。

好消息是，几乎可以确定客户端通过 TCP 与服务器交互，当 Backup 接管服务时，由于其状态与 Primary 相同，所以其也直到 TCP 连接的状态和 TCP 序列号，其生成回复报文时对应的序列号与 Primary 之前发送的是一样的，这样客户端就知道这是一个重复的报文，这样客户端就会丢弃该报文。

事实上，对于任何主从切换的复制系统，几乎不可能将系统设计成不产生重复输出，当主从切换时，有可能生成重复的输出，如果不是使用 TCP，我们就得使用其余的机制来处理。

假设 Primary 在崩溃前中断了与客户端的 TCP 连接，但 Backup 还是会有该 TCP 连接的信息，Backup 生成回复报文到客户端时，客户端会给 Backup 发送一个 TCP Reset，类似于 TCP error 的信息，但由于此时只有 Backup 一个副本，其可以任意处理，不用担心与其他副本有差异，事实上，Backup 会直接忽略这个报文。

## Test-and-Set 服务

如果 Primary 和 Backup 都在运行，但是它们之间的网络出现问题，同时它们又能各自与一些客户端通信，这时它们都会以为对方挂了，并且上线接管服务。此时它们都不向彼此发送 Log，Primary 和 Backup 同时在线，形成了脑裂（Split Brain）。此时需要第三方来决定 Primary 还是 Backup 允许上线，这个第三方服务就是 Test-and-Set 服务。

Test-and-Set 服务不运行在 Primary 和 Backup 的物理服务器上，VMware FT 需要通过网络支持 Test-and-Set 服务。这有点像一个锁，为了能够上线接管，Primary 和 Backup 或许会给 Test-and-Set 服务发送接管请求，此时第一个请求到达时，Test-and-Set 服务会说此时标志位为0，你可以接管，然后将0设置为1，下一个接管请求到达时，Test-and-Set 服务会说此时标志位为1，你不能接管。当网络出现故障，并且两个副本都认为对方已经挂了时，Test-and-Set 服务就是一个仲裁官，决定了两个副本中哪一个应该上线。即使没有网络分区，只要两个副本中任意一个觉得对方挂了，哪怕对方真的挂了，想要上线的那个副本仍然需要获得 Test-and-Set服务的锁。