The Design of a Practical System for Fault-Tolerant Virtual Machines

**一种实用的容错虚拟机系统的设计**

Daniel J. Scales, Mike Nelson, and Ganesh Venkitachalam

VMware, Inc

[{scales,mnelson,ganesh}@vmware.com](mailto:{scales,mnelson,ganesh}@vmware.com)

**摘要**

我们实现了一种商业的企业级系统，它提供容错虚拟机，基于通过在另一个服务器上的备份虚拟机复制主虚拟机（VM）执行的方法。我们在VMware vSphere 4.0上设计了一个完整的系统，它易于使用，运行在普通服务器上，并且通常减少实际应用的性能不到10%。此外，为了保持主和副虚拟机中几个实际应用步调一致，需要的数据带宽少于20Mbit/s，这使得在更长距离上实现容错成为可能。一个自动从失败之后恢复冗余的易于使用的商业系统，需要更多额外的组件，不仅仅是复制虚拟机的执行。我们设计并实现了这些额外的组件，在支持虚拟机运行商业应用的过程中，处理了很多实际问题。在这篇论文中，我们描述了我们的基本设计，讨论了不同设计和很多实现细节，并且提供了微基准测试和实际应用的性能报告。

# 简介

实现容错服务器的一种常见方法是主/备方法，如果主服务器故障，备份服务器总是可以接管。备份服务器的状态必须随时保持和主服务器几乎一致，这样在主服务器故障时，备份服务器才可以及时接管，这样故障对外部客户端是不可见的，并且没有数据丢失。在备份服务器上复制状态的一种方法是，把主服务器的所有状态，包括CPU、内存和I/O设备，全部连续不断的传递给备份服务器。然而，发送这个状态，特别是内存中的变化，需要非常大的带宽。

另外一种复制服务器的方法，使用更少的带宽，有时被称为状态机方法。这种方法的想法是把服务器当成确定的状态机，只要从相同的初始状态开始，保证它们以相同的顺序接收相同的输入请求，那么它们就会保持同步。因为很多服务器或者服务有一些操作不是确定的，需要一些额外的协调工作来使得主备保持同步。然而，保持主备同步的额外信息数量远小于主服务器上改变的状态数量（主要是内存更新）。

实现物理服务器上确定的执行的协调是困难的，尤其是处理器频率的提高。相反，运行在一个管理程序上的虚拟机（VM），是一个极好的实现状态机方法的平台。一个虚拟机可以被认为是一个定义良好的状态机，它的操作是被虚拟化的机器（包括所有的设备）的操作。跟物理服务器一样，虚拟机也有一些非确定性的操作（例如，读取当前时间或者传递一个中断），所以为了保持主备同步，额外的信息需要发送给备份虚拟机。因为管理程序完全控制虚拟机的执行，包括传递所有的输入，它可以捕获在主虚拟机上有关非确定性操作的所有必需信息，然后在备份虚拟机上正确的重放这些操作。

因此，状态机方法可以实现在运行普通硬件的虚拟机上，不需要修改任何硬件，对于最新的微处理器可以立即实现容错。另外，由于状态机方法需要较少的带宽，所以提供了大范围内主备的物理隔离可能性。例如，复制的虚拟机可以运行在分布于一个校园内的物理机器上，它比运行在同一建筑内的虚拟机更可靠。

在VMware vSphere 4.0平台中，我们使用主备方法实现了容错虚拟机，VMware vSphere 4.0以一种高效的方式运行完全虚拟化的x86虚拟机。因为VMware vSphere实现了一个完整的x86虚拟机，我们可以自动的对所有基于x86的操作系统和应用程序提供容错。能够记录主虚拟机的执行，并且确保在备份虚拟机上相同的执行，这种底层技术被称作确定性的重放。VMware vSphere容错（FT）基于确定性的重放技术，但是添加了创建一个完整的容错系统所必须的额外协议和功能。除了提供硬件的容错，我们的系统还自动在故障之后恢复冗余，通过在本地集群中任何可用的机器上开启一个新的备份虚拟机。到目前为止，确定性重放技术和VMware 容错仅支持单处理器的虚拟机。支持记录和重放多处理器虚拟机执行的工作目前还在进行中，目前遇到了严重的性能问题，因为几乎所有对共享内存的访问都是非确定性的操作。

Bressoud和Schneider [3] 描述了一个在HP PA-RISC平台上实现容错虚拟机的原型。我们的方法是类似的，但是因为性能的原因我们做了一些基础性的改变，并且研究了很多替代的设计。另外，为了创建一个完整的系统，它高效并且易于商业应用客户使用，我们必须设计和实现很多额外的系统组件，并处理一序列的实际问题。和很多其他被讨论的实际系统一样，我们仅尝试解决故障停止（fail-stop）故障[12]，它是一种服务器故障，在故障服务器引起一个外部可见的动作之前它是可被检测的。

论文的余下部分按照如下方式组织。首先，描述了我们的基本设计，详述了我们的基础协议，协议用于保证在主虚拟机故障之后备份虚拟机接管的过程中没有数据丢失。其次，我们详细介绍了，为了创建一个健壮的、完整的和自动化的系统，所需要被处理的很多实际问题。我们还描述了几个不同的用于实现容错虚拟机的设计选择，讨论了这些选择的折衷。然后，我们提供了基于我们的实现的一些基准和商业应用的性能测试结果。最后，我们描述了相关工作并总结。

# 基本容错（FT）设计

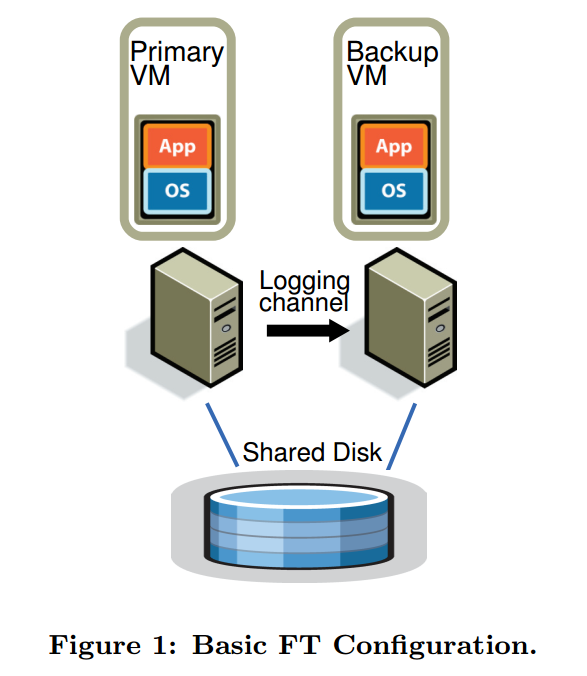


图1展示了我们的容错虚拟机系统的基本配置。对于一个给定的虚拟机，当我们想提供容错（主虚拟机），我们在另外一台物理服务器上运行一个备份虚拟机，它用于保持同步，并且执行和主虚拟机一样的操作，虽然有一个很小时间的延迟。我们称这两个虚拟机处于虚拟一致。这些虚拟机的虚拟磁盘都在共享存储设备上，因此输入输出都可以被主备虚拟机访问。（在4.1章节我们将会讨论一种采用分离不共享虚拟磁盘的主备虚拟机方案）只有主虚拟机在网络中广告它的存在，所以所有的网络输入都会到达主虚拟机。相似地，所有其他的输入（例如键盘和鼠标）也只到达主虚拟机。

主虚拟机接收的所有输入，通过一个叫日志通道的网络连接，被发送给备份虚拟机。对于服务器工作负载，主要的输入流量是网络和磁盘。额外的信息，如在下面2.1章节讨论的，根据需要传输给备份虚拟机，以确保备份虚拟机和主虚拟机以相同的方式执行非确定性的操作。结果是备份虚拟机总是和主虚拟机执行一样。然而，管理程序丢弃备份虚拟机的输出，所以只有主虚拟机产生实际的输出返回给客户端。如2.2章节所述，主备虚拟机遵循一个特定协议，包括备份虚拟机的显式确认，为了确保在主虚拟机故障的情况下没有数据丢失。

为了检测主或者备虚拟机是否故障了，我们的系统使用了一个相关服务器之间的心跳和日志通道的流量监控的组合。另外，我们必须确保只有一个主或者备份虚拟机在接管执行，即使是在一个裂脑（split-brain）的处境下，其中主备服务器相互之间失去了通信。

在接下的章节中，我们提供了几个重要领域的更多细节。在2.1章节，我们介绍了一些有关确定性重放技术的细节，它通过日志通道传递的信息，确保了主备虚拟机保持同步。在2.2章节，我们描述了容错协议的一个基础规则，它确保在主虚拟机故障的情况下没有数据丢失。在2.3章节，我们描述了如何正确的故障检测和响应方法。

## 确定性重放的实现

正如我们已经提到的，复制服务器（或者虚拟机）的执行可以类比于一个确定性状态机的复制。如果两个确定性状态机从相同的初始条件开始，以相同的顺序提供完全相同的输入，那么它们将会走过相同序列的状态，产生相同的输出。虚拟机有非常广泛的输入，包括进来的网络包、读磁盘和来自键盘和鼠标的输入。非确定性事件（例如虚拟中断）和非确定性操作（例如读处理器的时钟周期计数）也会影响虚拟机的状态。这对于运行在任何操作系统和工作负载下的任何虚拟机执行的复制提出了三个挑战：（1）正确捕获所有输入和非确定性因素，以确保备份虚拟机的确定性执行。（2）把所有输入和非确定性因素正确地应用到备份虚拟机中。（3）这种方式不能降低性能。另外，x86微处理器中很多复杂的指令有未定义副作用，因此是非确定性的。捕获这些未定义的副作用，重放它们产生相同的状态，提出了一个额外的挑战。

VMware确定性重放 [15] 正好为VMware vSphere平台上的x86虚拟机，提供了这个功能。确定性重放记录，虚拟机的输入和跟虚拟机执行有关的所有可能的非确定性因素，把它们以日志条目流的形式写入到一个日志文件。随后，通过读取日志文件中日条目，虚拟机的整形可以被正确的重放出来。对于非确定性操作，足够的日志信息允许操作被重放，产生相同的状态变化和输出。对于非确定性事件，例如定时器或者IO完成中断，事件发生时的指令也被记录。在重放过程中，在指令流中相同的点，事件被传递出来。VMware确定性重放实现了一种有效的事件记录和事件传送的机制，它采用了多方面的技术，包括使用了和AMD[2]和Intel[8]联合开发的硬件性能计数器。

Bressoud 和Schneider [3]提到把虚拟机的执行划分成不同时期，其中非确定性事件，例如中断，仅在一个时期的结尾被传送。时期的概念意味着一个批量的机制，因为在指令确切发生的地方传送每一个中断非常昂贵。然而，我们的事件传送机制足够高效，所以VMware确定性重放没必要使用时期。每一个中断在它发生时都被记录，然后在重放时，在恰当的指令中被高效的传送出来。

## 容错（FT）协议

对于VMware容错，我们使用确定性重放产生必须的日志条目，用来记录主虚拟机的执行，但是不把日志条目写到磁盘，而是通过日志通道把它们发送给备份虚拟机。备份虚拟机实时的重放这些日志条目，因此和主虚拟机执行一样。然而，我们必须在日志通道上增加严格容错协议有关的日志条目，它的目的是为了确保我们能实现容错。我们的基本要求如下：

**输出要求**：如果备份虚拟机在主虚拟机故障之后接管，那么备份虚拟机将会以跟主虚拟机发送到外部世界的全部输出完全一致的方式继续执行。

注意到在一个失效备援发生之后（例如，备份虚拟机在主虚拟机故障之后接管），备份虚拟机很可能将会，以跟主虚拟机曾经非常不同的方式开始执行，因为在执行过程中很多非确定性事件的发生。然而，只要备份虚拟机满足输出要求，在失效备援到备份虚拟机的过程中，没有外部可见的状态或者数据丢失，那么客户端在它们的服务中，将不会注意到中断或者非一致性。

输出要求可以这样被确保：延迟任何外部输出（典型的是一个网络包），直到备份虚拟机接收到所有足以让它重放执行到输出操作点相关的信息。一个必要条件是备份虚拟机必须接收到输出操作之前产生的所有日志条目。这些日志条目将允许它执行到最后日志记录的点。然而，假设在主虚拟机执行了输出操作之后，立即发生了一个故障。备份虚拟机必须知道它必须重放到输出操作的点，并且只在那个点“上线”（停止重放，并且接管作为主虚拟机，如2.3章节所描述的）。如果备份虚拟机在输出操作之前的最后一个日志条目点上线，一些非确定性事件（例如，传送给虚拟机的定时器中断），在它执行输出操作之前，可能改变它的执行路径。

在以上给出的约束下，最简单的达到输出要求的方法是，对于每一个输出操作创建一条特殊的日志条目。然后，输出要求可以被下面的这条特殊规则达到：

**输出规则**：主虚拟机，直到备份虚拟机接收并确认了跟产生输出操作关联的日志条目之前，不会发送输出到外部世界。

如果备份虚拟机接收到所有日志条目，包括输出产生操作的那条日志条目，那么，备份虚拟机将能够准确的重新产生主虚拟机在输出点的状态，所以如果主虚拟机挂了，备份虚拟机将能够正确的达到跟输出一致的状态。相反，如果备份虚拟机没有接收到所有必须的日志条目就接管了，那么它的状态可能快速偏离到跟主虚拟机输出不一致。输出规则在某些方面类似于在[11]中描述的方法，其中一个“外部同步”IO可以被缓冲，只要在下一次外部通信之前确实写到磁盘中。

注意输出规则没有任何关于停止主虚拟机执行的说明。我们只需要延迟发送输出，至于虚拟机本身可以继续执行。因为操作系统是通过异步中断来标识非阻塞的网络和磁盘输出的完成，虚拟机可以轻松的继续执行，不会被输出的延迟立即影响。相反，以前的工作[3,9]明确指出虚拟机在输出之前必须完全停止执行，直到备份虚拟机确认了所有来自主虚拟机的必要信息。

作为一个例子，在图2中我们以图的形式展示了容错协议的要求。这个图展示了在主和备虚拟机中事件的时间轴。从主线到备份线的箭头反应了日志条目的转移，从备份线到主线从的箭头反应了日志条目的确认。关于异步事件、输入和输出操作相关的信息必须确保以日志条目的形式发送到备份虚拟机。如图中所示，一个外部世界的输出被延迟，直到主虚拟机收到备份虚拟机对输出操作关联的日志条目的确认。如果输出规则被遵循，备份虚拟机将能够以跟主虚拟机最后一个输出相一致的状态接管。

我们不能保证在一个失效备援的情况下，所有的输出恰好只被产生一次。在不使用两阶段提交事务的前提下，当主虚拟机打算发送一个输出时，备份虚拟机没有办法知道，主虚拟机是在发送最后一个输出之前或者之后故障了。幸运的是，网络基础设施（包括常用的TCP）被设计成可以处理丢包和相同（重复）包。注意到在主虚拟机故障时，输入到主虚拟机的网络包可能丢失，因此也不会传送到备份虚拟机。然而，输入的网络可能因为各种跟服务器故障无关的原因被丢失，所以网络基础设施、操作系统和应用程序都被编码成它们可以处理网络包丢失。

## 故障检测与响应

如以上所提到的，如果其他虚拟机好像故障了，主和备份虚拟机必须快速响应。如果备份虚拟机故障了，主虚拟机将继续在线 --- 意味着，退出记录模式（因此停止在日志通道中发送日志条目），开始正常执行。如果主虚拟机故障了，备份虚拟机应该以类似的方式在线，但是处理过程有一些复杂。因为执行的延迟，备份虚拟机可能由很多接收并确认的日志条目并没有消费掉，因为备份虚拟机还没有达到恰当的执行点。备份虚拟机必须继续重放日志条目的执行，直到它消费了最后一个日志条目。这个时候，备份虚拟机将会停止重放模式，然后像一个正常虚拟机一样开始执行。本质上，备份虚拟机被提升为主虚拟机（现在它缺少一个备份虚拟机）。因为它不再是备份虚拟机，当客户操作系统需要输出操作时，新的主虚拟机将会对外部世界产生输出。在向正常模式转移的过程中，可能需要一些设备专用的操作来让输出恰当的发生。特别是，为了网络传输的目的，VMware容错会自动在网络中把新主虚拟机的网卡地址广而告之，这样物理网络交换机将会知道新主虚拟机的地址。另外，新提升的主虚拟机可能需要重新发出一些磁盘IO（如3.4章节所述）。

有很多方法可以拿来尝试检测主和备份虚拟机的故障。VMware容错使用运行容错虚拟机服务器之间的UDP心跳，来检测是否由服务器故障了。另外，VMware容错监控主虚拟机发送到备份虚拟机的日志消息和备份虚拟机发送到主虚拟机的确认消息。因为定期的定时器中断，日志流量必须是定期的，并且对于一个运行的客户操作系统来说从来不会停止。因此，一个日志条目或者确认消息流的停止就标识了虚拟机的一个故障。如果心跳或者日志流量停止了超过一定时间（以几秒钟的顺序），则故障被声明发生了。

然而，任何这样的故障检测方法都容易受到分裂脑问题的影响。如果备份服务器停止收到来自主服务器的心跳，那可能标志着主服务器可能故障了，或者可能意味着任然在运行的服务器之间的全部网络连接都丢失了。如果在主虚拟机实际上还在运行的情况下，备份虚拟机上线，这可能会导致数据损坏或者客户端和虚拟机之间的通信问题。因此，当故障被检测到时，我们必须确保主或者备份虚拟机中只有一个能上线。为了避免分裂脑问题，我们使用了存储虚拟机磁盘的共享存储设备。当主或者备份虚拟机将要上线时，它会在共享存储设备上执行一个test-and-set操作。如果操作成功了，虚拟机就会被允许上线。如果操作失败了，其他虚拟机已经上线了，所以当前虚拟机就停止自己（“提交终止自己”）。当虚拟机尝试做这个原子操作时不能访问共享存储设备，就继续等待直到能访问。注意到，如果因为存储网络的一些故障导致共享存储不能被访问，那么虚拟机就可能做不了任何有意义的工作，因为虚拟磁盘是放在这些共享存储上。因此，采用共享存储来解决分裂脑问题，不会引入任何额外的不可用性。

设计的最后一个方面是一旦故障发生了，其中一个虚拟机上线了，VMware容错会自动通过在另外一个主机上开启一个新的备份虚拟机来恢复冗余。虽然这个过程在以前很多工作中并没有被涉及到，但是对于使容错虚拟机有用是基本条件，并且它需要仔细的设计。更多的细节在3.1章节中给出。

# 容错（FT）的实际实现

第2章描述了容错的基础设计和协议。然而，为了创建可用的、健壮的自动化系统，还必须设计和实现很多其他的组件。

## 容错虚拟机（FT VM）的启动和重启

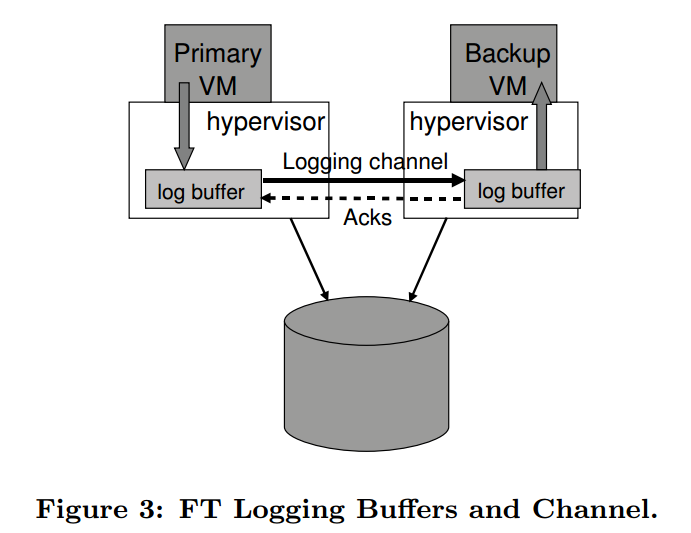
必选设计的最大的附加组件之一是，启动一个跟主虚拟机状态相同的备份虚拟机的机制。这个机制也被用于在故障发生之后重新启动一个备份虚拟机。因此，对于一个处于任意状态（也就是不仅仅是启动）的运行中的主虚拟机，这个机制必须可用。另外，我们更希望这个机制不会显著的破坏主虚拟机的执行，因为那样会影响虚拟机当时所有的客户端。

对于VMware容错，我们适配了VMware vSphere中的VMotion功能。VMware VMotion可以把一个运行中的虚拟机从一个服务器迁移到另外一个服务器，影响非常小 --- 虚拟机停止的时间通常少于一秒钟。我们创建了VMotion 的一个修改形式，它可以在远程服务器上拷贝一个运行的虚拟机，并且不会破坏本地服务器上的虚拟机。那就是，我们修改的容错VMotion在远程服务器上拷贝一个虚拟机，而不是迁移它。容错VMotion也会建立一个日志通道，使得源虚拟机进入日志模式，充当主虚拟机，而目标虚拟机进入重放模式，充当备份虚拟机。就像正常的VMotion，容错VMotion中断主虚拟机的执行少于一秒钟。因此，对于运行的虚拟机，启用容错是一个简单的，并且非破坏性的操作。

启动一个备份虚拟机的另外一个方面是如何选择一个服务器。容错虚拟机运行在共享存储的服务器集群中，所以所有虚拟机可以运行在集群中任何一台服务器上。这个灵活性允许VMware vSphere在一台或者多台服务器故障的时候，恢复容错冗余。VMware vSphere实现了一个维护管理和资源信息的集群服务。当故障发生时，主虚拟机需要一个新备份虚拟机来重新建立冗余时，主虚拟机通知集群服务它需要一个新备份虚拟机。集群服务依据资源的使用和其他限制情况，选择最优的适合运行备份虚拟机的服务器，然后调用一个容错VMotion来创建新的备份虚拟机。结果是在服务器故障几分钟之内，VMware容错就能够重新建立虚拟机冗余，所有这些都不需要明确中断容错虚拟机的执行。

## 日志通道的管理

在管理日志通道的流量中由很多有趣的实现细节。在我们的实现中，管理程序为主和备份虚拟机维护了一个很大的日志条目缓冲区。当主虚拟机执行时，它向日志缓冲区中生产日志条目，类似的，备份虚拟机从日志缓冲去中消费日志条目。主虚拟机日志缓冲的内容会尽可能快地刷到日志通道中，到达日志通道中的日志条目会被立即读取到备份虚拟机的日志缓冲中。当备份虚拟机每次从网络中读取到一些日志条目到自己的缓冲中，它会向主虚拟机返回确认。这些确认使得VMware容错可以决定，根据输出规则延迟的输出可以被发送出去。图3展示了这个过程。



如果备份虚拟机读取下一条日志条目时，遇到了一个空日志缓冲，它会停止执行，直到出现一条新的日志条目。因为备份虚拟机不跟外部通信，所以这个暂停不会影响到虚拟机的任何客户端。类似的，当主虚拟机需要写一条日志条目时，遇到了一个满日志缓冲，它必须停止执行，直到日志条目可以被刷出。这个停止执行是一个自然的流程控制机制，当主虚拟机以过快的速度生产日志条目时，它减慢了主虚拟机。然而，这个暂停会影响到虚拟机的客户端，因为主虚拟机将会完全停止并且无反应，直到它可以写日志条目为止才继续执行。因此，我们的实现必须被设计成最小化主虚拟机日志缓冲填满的可能性。

当主虚拟机日志缓冲填满时，其中一个原因是备份虚拟机执行得太慢，因此消费日志条目太慢。一般而言，备份虚拟机必须能够以跟主虚拟机记录执行大致相同的速度来重放执行过程。幸运的是，在VMware确定性重放中记录和重放的开销大致上是相同的。然而，如果备份虚拟机所在的服务器加载了过多其他的虚拟机（因此就过度使用资源），备份虚拟机就有可能因为拿不到足够的CPU和内存资源，导致不能像主虚拟机执行一样快，尽管管理程序的虚拟机调度器做了最大的努力。

避免当日志缓冲填满时出现意外停顿，存在另外一个原因导致我们不希望执行延迟变得太大。当主虚拟机故障时，在备份虚拟机上线和开始与外部世界通信之前，它必须通过重放已经确认过的日志条目来“赶上”。在故障发生点完成重放的事件基本上就是执行延迟，所以备份虚拟机上线的事件大致上等于故障检测时间加上当前的执行延迟时间。因此，我们不希望执行延迟时间太大（超过1秒），因为那样会让故障转移时间增加更多。

因此，我们有一个额外的机制来减慢主虚拟机，防止备份虚拟机落后太多。在我们发送和确定日志条目的协议中，我们会发送额外的信息，用来确定主和备份虚拟机之间实时的执行延迟。通常执行延迟少于100毫秒。如果备份虚拟机开始有一个显著的执行延迟（例如，超过1秒），VMware容错就开始通知调度器少给一些CPU（最初只有百分之几），来减慢主虚拟机。我们使用一个慢反馈机制，它会尝试逐渐的精确控制CPU以允许备份虚拟机来匹配它的执行。如果备份虚拟机继续落后，我们继续逐渐的减少主虚拟机的CPU。相反的，如果备份虚拟机赶上了，我们会逐渐的增加主虚拟机的CPU，直到备份虚拟机回到只有轻微的延迟为止。

注意到这样对主虚拟机的减速是非常少的，通常只会发生在系统在极端压力下。地5章节所有的性能数据都包括了这样的减速。

## 容错虚拟机（FT VM）的操作

另外一个实际的事情是处理各种会被应用到主虚拟机的控制操作。例如，如果主虚拟机被明确的关机了，备份虚拟机也应该被停止，并且不能企图上线。另外一个例子，主虚拟机上任何资源管理的改变（例如增强的CPU共享）也应该被应用到备份虚拟机上。对于这些操作，特别的日志条目会通过日志通道从主虚拟机发送到备份虚拟机，为了在备份虚拟机上实现恰当的操作。

通常来说，虚拟机大部分操作应该只在主虚拟机上开始，然后VMware容错会发送任何必须的控制条目，以引起备份虚拟机上恰当的改变。在主和备份虚拟机上独立完成的操作只有VMotion。那是，主和备份虚拟机可以独立的被迁移到其他主机上。注意VMware容错会保证没有虚拟机会被迁移到其他虚拟机所在的服务器上，因为那样会导致不再提供容错。

主虚拟机的迁移比正常的迁移增加了一些复杂性，因为备份虚拟机必须和源主虚拟机断开，然后在恰当的时间重新连接到目标主虚拟机。备份虚拟机的迁移有一个类似的问题，但是增加了额外的复杂性。对于一个正常的虚拟机迁移，我们要求所有未决的磁盘IO静默（也就是说完成的）就像在迁移发生时最后的切换一样。对于一个主虚拟机，这种静默是易于处理的，只要等到物理IO完成并且通知到虚拟机为止。然而，对于一个备份虚拟机，没有简单的方法可以让所有的IO在必须的点上完成，因为备份虚拟机必须重放主虚拟机的执行，然后在相同点上完成IO。主虚拟机可能运行一个在正常执行时总是有磁盘IO的工作负载。VMware容错有一个唯一的方法来解决这个问题。当备份虚拟机处于迁移的最后切换点，它通过日志通道请求主虚拟机临时及静默所有IO。备份虚拟机的IO自然就会静默，就像单个执行点一样，因为它会重放主虚拟机的静默操作。

## 磁盘IO的实现问题

关于磁盘IO，存在一序列微妙的实现问题。首先，由于磁盘操作是非阻塞的，所以可以并行执行，同时访问磁盘相同位置的磁盘操作会导致非确定性。而且，我们磁盘IO的实现是在虚拟机的内存中直接使用DMA，所以相同的磁盘操作即是访问相同内存页也会导致非确定性。我们的解决方案是检测这样的磁盘IO竞争（这些很少发生），然后强制性的让这些竞争磁盘操作在主和备份虚拟机中以相同的方式顺序执行。

其次，虚拟机中磁盘操作也会和应用程序（或者操作系统）的内存访问产生竞争，因为磁盘操作是通过DMA直接访问虚拟机的内存。例如，当虚拟机的应用或者操作系统读取一个内存块，同时一个磁盘读取也发生在同样的内存块上，这会产生一个非确定性的结果。这种情况很少发生，但是我们必须检测它，并在它发生时处理它。一个解决方案是对于磁盘操作的内存页临时设置页保护。当虚拟机碰巧访问一个未决的磁盘操作的内存页时，页保护会导致一个陷阱，这时虚拟机可以暂停，直到磁盘操作完成。因为改变这些内存页上的MMU保护是一个昂贵的操作，所以我们选择使用弹性缓冲区。一个弹性缓冲区是一个跟磁盘操作内存大小相同的临时缓冲区。一个磁盘读取操作被修改成读取弹性缓冲区中特定的数据 ，只有在IO完成之后数据才会被拷贝到客户内存中。类似的，对于一个磁盘写入操作，发送的数据首先被写入弹性缓冲区，磁盘写入操作被修改成从弹性缓冲区写入数据。弹性缓冲区的使用会减慢磁盘操作，但是我们没有发现任何显著的性能损失。

第三，当主虚拟机故障，备份虚拟机接管时，有一些问题是跟主虚拟机中未决的（也就是说未完成的）磁盘IO有关。没有办法让新提升的主虚拟机明确这些磁盘IO被发到磁盘或者成功完成。另外，因为在备份虚拟机上这些磁盘IO没有对外发布，当新提升的主虚拟机继续执行时将不会有显式的IO完成，它有可能引起虚拟机中客户操作系统开始一个退出或者重置的程序。我们可以发送一个错误的完成来标志这些IO失败了，因为即使IO成功完成返回错误也是可接受的。然而，客户操作系统从它的本地磁盘可能不会正确的响应这些错误。相反，在备份虚拟机上线的过程中，我们重新发布这些未决的IO。因为我们消除了所有的竞争，并且所有的IO都直接制定了访问那块内存和磁盘块，这些磁盘操作可以被重新发布，即使它们已经成功完成（也即是说他们是幂等的）。

## 网络IO的实现问题

# 替代设计

## 共享磁盘 vs. 非同享磁盘

## 备份虚拟机上执行磁盘读取

# 性能评估

## 基本性能结果

## 网络基准测试

# 相关工作

# 结论与未来工作

**致谢**

# 参考