容错虚拟机实用系统设计

Daniel J. Scales, Mike Nelson 和 Ganesh Venkitachalam
VMware, Inc
{scales,mnelson,ganesh}@vmware.com

摘要

我们已经实现了一个商业企业级系统，用于提供容错虚拟机，该系统基于通过在另一台服务器上的备份虚拟机复制主虚拟机 (VM) 的执行的方法。我们已经在 VMware vSphere 4.0 中设计了一个完整的系统，该系统易于使用，可在商用服务器上运行，并且通常将实际应用程序的性能降低不到 10%。此外，对于几个实际应用程序，保持主虚拟机和辅助虚拟机同步执行所需的数据带宽小于 20 Mbit/s，这使得可以在更长的距离上实现容错。一个易于使用的商业系统，可以在发生故障后自动恢复冗余，除了复制的虚拟机执行之外，还需要许多额外的组件。我们已经设计并实现了这些额外的组件，并解决了在支持运行企业应用程序的虚拟机时遇到的许多实际问题。在本文中，我们描述了我们的基本设计，讨论了替代设计选择和一些实现细节，并提供了微基准测试和实际应用程序的性能结果。

1. 简介

实现容错服务器的一种常见方法是主/备方法 [1]，其中备用服务器始终可用，以便在主服务器发生故障时接管。备用服务器的状态必须始终与主服务器的状态几乎相同，以便备用服务器可以在主服务器发生故障时立即接管，并且以这种方式，故障对外部客户端隐藏并且没有数据丢失。在备用服务器上复制状态的一种方法是将更改发送到主服务器的所有状态，包括 CPU、内存和 I/O 设备，几乎连续地发送到备用服务器。但是，发送此状态所需的带宽，特别是内存中的更改，可能非常大。

复制服务器的另一种方法可以使用更少的带宽，有时被称为状态机方法 [13]。其思想是将服务器建模为确定性状态机，这些状态机通过从相同的初始状态启动并确保它们以相同的顺序接收相同的输入请求来保持同步。由于大多数服务器或服务都有一些非确定性操作，因此必须使用额外的协调来确保主服务器和备用服务器保持同步。但是，保持主服务器和备用服务器同步所需的额外信息量远小于主服务器中正在更改的状态量（主要是内存更新）。

实现协调以确保物理服务器的确定性执行 [14] 很困难，尤其是在处理器频率增加时。相比之下，在 hypervisor 之上运行的虚拟机 (VM) 是实现状态机方法的绝佳平台。虚拟机可以被认为是一个定义明确的状态机，其操作是被虚拟化的机器的操作（包括其所有设备）。与物理服务器一样，虚拟机也有一些非确定性操作（例如，读取时间时钟或传递中断），因此必须将额外的信息发送到备份，以确保它保持同步。由于 hypervisor 完全控制虚拟机的执行，包括传递所有输入，因此 hypervisor 能够捕获有关主虚拟机上非确定性操作的所有必要信息，并在备份虚拟机上正确重放这些操作。

因此，状态机方法可以在通用硬件上的虚拟机上实现，无需任何硬件修改，从而可以立即为最新的微处理器实现容错。此外，状态机方法所需的低带宽允许主服务器和备份服务器之间更大的物理分离的可能性。例如，复制的虚拟机可以在分布在整个校园的物理机上运行，这比在同一建筑物中运行的虚拟机提供更高的可靠性。

我们已经在 VMware vSphere 4.0 平台上使用主/备方法实现了容错虚拟机，该平台以高效的方式运行完全虚拟化的 x86 虚拟机。由于 VMware vSphere 实现了完整的 x86 虚拟机，因此我们能够自动为任何 x86 操作系统和应用程序提供容错。允许我们记录主服务器的执行并确保备份服务器以相同方式执行的基础技术被称为确定性重放 [15]。VMware vSphere 容错 (FT) 基于确定性重放，但添加了必要的额外协议和功能来构建完整的容错系统。除了提供硬件容错之外，我们的系统还会在发生故障后通过在本地集群中的任何可用服务器上启动新的备份虚拟机来自动恢复冗余。目前，确定性重放和 VMware FT 的生产版本仅支持单处理器虚拟机。记录和重放多处理器虚拟机的执行仍在进行中，由于几乎每次访问共享内存都可能是不确定的操作，因此存在严重的性能问题。

Bressoud 和 Schneider [3] 描述了 HP PA-RISC 平台的容错虚拟机的原型实现。

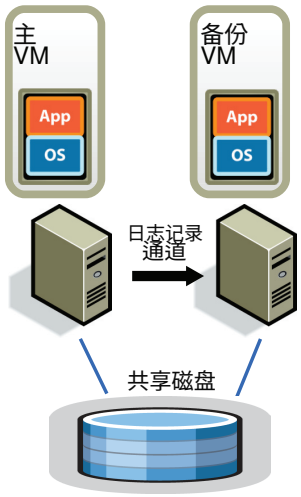

图 1：基本 FT 配置。

HP PA-RISC 平台的容错虚拟机的实现。我们的方法类似，但出于性能原因，我们进行了一些根本性的更改，并研究了许多设计方案。此外，我们还必须设计和实现系统中的许多其他组件，并处理许多实际问题，以构建一个完整的系统，该系统对于运行企业应用程序的客户来说是高效且可用的。与讨论的大多数其他实际系统类似，我们只尝试处理故障停止故障 [12]，这些故障是在发生故障的服务器导致不正确的外部可见操作之前可以检测到的服务器故障。

本文的其余部分组织如下。首先，我们描述我们的基本设计并详细说明我们的基本协议，以确保在备份 VM 在主 VM 发生故障后接管时，不会丢失任何数据。然后，我们详细描述了构建一个健壮、完整和自动化的系统必须解决的许多实际问题。我们还描述了实现容错 VM 时出现的几种设计选择，并讨论了这些选择中的权衡。

接下来，我们将给出我们实现的性能结果，
适用于一些基准测试和一些实际的企业应用程序。
最后，我们将描述相关工作并得出结论。

2. 基本 FT 设计

图 1 显示了我们的容错虚拟机系统的基本设置。对于我们希望提供容错的给定虚拟机（主虚拟机），我们在不同的物理服务器上运行一个备份虚拟机，该虚拟机保持同步并以与主虚拟机相同的方式执行，尽管存在较小的时间延迟。我们说这两个虚拟机处于虚拟锁定状态。虚拟机的虚拟磁盘位于共享存储上（例如光纤通道或 iSCSI 磁盘阵列），因此主虚拟机和备份虚拟机可以访问输入和输出。（我们将在第 4.1 节中讨论主虚拟机和备份虚拟机具有单独的非共享虚拟磁盘的设计。）只有主虚拟机在网络上公布其存在，因此所有网络输入都进入主虚拟机。

同样，所有其他输入（例如键盘和鼠标）都只发送到主虚拟机。

主虚拟机接收到的所有输入都将发送到备份虚拟机通过一个称为日志通道的网络连接。对于服务器工作负载，主要的输入流量是网络和磁盘。如下面第 2.1 节中所述，会根据需要传输其他信息，以确保备份虚拟机以与主虚拟机相同的方式执行非确定性操作。结果是备份虚拟机始终与主虚拟机以相同的方式执行。

但是，备份虚拟机的输出被虚拟机监控程序丢弃，因此只有主虚拟机产生返回给客户端的实际输出。如第 2.2 节所述，主虚拟机和备份虚拟机遵循特定的协议，包括备份虚拟机的显式确认，以确保如果主虚拟机发生故障，不会丢失任何数据。

为了检测主虚拟机或备份虚拟机是否发生故障，我们的系统使用相关服务器之间的心跳检测以及对日志通道上的流量的监控相结合的方式。
此外，我们必须确保即使在主服务器和备份服务器彼此失去通信的脑裂情况下，也只有主虚拟机或备份虚拟机接管执行。

在以下章节中，我们将提供关于几个重要领域的更多细节。在第2.1节中，我们将详细介绍确定性重放技术，该技术通过日志通道发送的信息来确保主虚拟机和备份虚拟机保持同步。在第2.2节中，我们将描述我们的FT协议的一个基本规则，该规则确保如果主虚拟机发生故障，则不会丢失任何数据。在第2.3节中，我们将描述我们用于以正确方式检测和响应故障的方法。

2.1 确定性重放实现

正如我们所提到的，复制服务器（或虚拟机）的执行可以建模为确定性状态机的复制。如果两个确定性状态机以相同的初始状态启动，并以相同的顺序提供完全相同的输入，那么它们将经历相同的状态序列并产生相同的输出。虚拟机具有广泛的输入，包括传入的网络数据包、磁盘读取以及来自键盘和鼠标的输入。非确定性事件（如虚拟中断）和非确定性操作（如读取处理器的时钟周期计数器）也会影响虚拟机的状态。这为复制运行任何操作系统和工作负载的任何虚拟机的执行带来了三个挑战：

（1）正确捕获所有输入和非确定性，以确保备份虚拟机的确定性执行；（2）正确地将输入和非确定性应用于备份虚拟机；（3）以不降低性能的方式执行此操作。此外，x86微处理器中的许多复杂操作都具有未定义的，因此是非确定性的副作用。捕获这些未定义的副作用并重放它们以产生相同的状态提出了额外的挑战。

VMware 确定性重放 [15] 为 VMware vSphere 平台上的 x86 虚拟机提供了完全相同的功能。确定性重放记录虚拟机的输入以及与虚拟机执行相关的所有可能的非确定性，并将其写入到日志文件的日志条目流中。通过从文件中读取日志条目，可以稍后精确地重放虚拟机执行。对于非确定性操作，会记录足够的信息，以允许以相同的状态更改和输出来重现该操作。对于诸如计时器或 IO 完成之类的非确定性事件-

中断，还会记录事件发生的精确指令。在重放期间，事件在指令流中的同一点传递。VMware 确定性重放实现了一种高效的事件记录和事件传递机制，该机制采用了各种技术，包括使用与 AMD [2] 和 Intel [8] 联合开发的硬件性能计数器。

Bressoud 和 Schneider [3] 提到将虚拟机的执行划分为多个时期（epoch），其中诸如中断之类的非确定性事件仅在时期结束时传递。时期（epoch）的概念似乎被用作一种批处理机制，因为在事件发生的精确指令处单独传递每个中断的成本太高。但是，我们的事件传递机制足够高效，以至于 VMware 确定性重放无需使用时期（epoch）。每个中断都会在其发生时被记录下来，并在重放时在适当的指令处有效地传递。

2.2 FT 协议

对于 VMware FT，我们使用确定性重放来生成必要的日志条目以记录主虚拟机的执行，但是我们不是将日志条目写入磁盘，而是通过日志通道将其发送到备份虚拟机。

备份虚拟机实时重放条目，因此执行方式与主虚拟机相同。但是，我们必须在日志通道上使用严格的 FT 协议来增强日志条目，以确保实现容错。我们的基本要求如下：

> 输出要求：如果备份 VM 在主 VM 发生故障后接管，则备份VM 将继续执行，其方式与主VM 发送到外部世界的所有输出完全一致。

请注意，在故障转移发生后（即备份 VM 在主 VM 发生故障后接管），备份 VM很可能会以与主 VM 继续执行的方式截然不同的方式开始执行，因为执行期间会发生许多非确定性事件。

但是，只要备份 VM 满足输出要求，在故障转移到备份 VM 期间，就不会丢失任何外部可见的状态或数据，并且客户端不会注意到其服务中的任何中断或不一致。

可以通过延迟任何
外部输出（通常是网络数据包）来确保输出要求，直到备份 VM 收到所有信息，这些信息将允许它至少重放
执行到该输出操作的点。
一个必要的条件是备份 VM 必须收到在输出操作之前生成的所有日志条目。这些日志条目将允许它执行到最后一个日志条目的点。但是，假设在主服务器执行输出操作后立即发生故障。备份 VM 必须知道它必须保持重放到输出操作的点，并且只能在该点“上线”（停止重放并接管作为主VM，如第 2.3 节中所述）。如果备份要在输出操作之前的最后一个日志条目的点上线，则某些非确定性事件（例如，传递到 VM 的计时器中断）可能会在它执行输出操作之前更改其执行路径。

鉴于上述约束，强制执行
输出要求的最简单方法是在以下位置创建一个特殊的日志条目

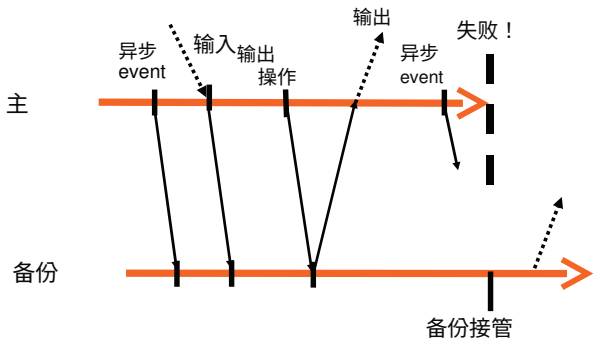

图 2：FT 协议。

每个输出操作。然后，输出要求可能
由这个特定规则强制执行：

> 输出规则：主 VM 在备份 VM 接收并确认与生成输出的操作关联的日志条目之前，不得向外部世界发送输出。

如果备份 VM 已经接收到所有日志条目，包括输出生成操作的日志条目，那么备份 VM 将能够准确地重现该输出点的主 VM 的状态，因此如果主 VM 崩溃，备份将正确地达到与该输出一致的状态。相反，如果备份 VM在没有接收到所有必要的日志条目的情况下接管，那么它的状态可能会迅速发散，以至于与主 VM 的输出不一致。输出规则在某些方面类似于 [11] 中描述的方法，其中“外部同步”IO 实际上可以被缓冲，只要它在下一次外部通信之前实际写入磁盘即可。

请注意，输出规则没有说明停止主 VM 的执行。我们只需要延迟输出的发送，但 VM 本身可以继续执行。由于操作系统使用异步中断进行非阻塞网络和磁盘输出，以指示完成，因此 VM 可以轻松地继续执行，并且不一定会立即受到输出延迟的影响。相比之下，之前的工作 [3, 9] 通常表明主 VM 必须完全停止才能执行输出，直到备份 VM 确认来自主 VM 的所有必要信息。

例如，我们展示了一个图表，说明了图 2 中 FT 协议的要求。该图显示了
主 VM 和备份 VM 上的事件时间线。从
主线到备份线的箭头表示
日志条目的传输，从
备份线到主线的箭头表示
确认。有关异步事件、输入和
输出操作的信息必须作为日志条目
发送到备份并进行确认。如图所示，到
外部世界的输出被延迟，直到主 VM 收到
来自备份 VM 的确认，表明它已
收到与输出操作关联的日志条目。
鉴于遵循了输出规则，备份 VM 将
能够以与主 VM 的

上次输出一致的状态接管。

我们无法保证在故障转移情况下所有输出都只产生一次。如果不使用具有两阶段提交的事务，当主服务器打算发送输出时，备份服务器无法确定主服务器是在发送最后一个输出之前还是之后立即崩溃的。幸运的是，网络基础设施（包括 TCP 的常见使用）旨在处理丢失的数据包和相同（重复）的数据包。请注意，传入主服务器的数据包也可能在主服务器发生故障期间丢失，因此不会传递到备份服务器。但是，传入的数据包可能会因与服务器故障无关的任何原因而被丢弃，因此网络基础设施、操作系统和应用程序都编写为确保它们可以补偿丢失的数据包。

2.3 检测和响应故障

如上所述，如果另一个 VM 似乎已发生故障，则主 VM 和备份 VM 必须快速响应。如果备份 VM 发生故障，则主 VM 将上线 – 也就是说，离开记录模式（因此停止在日志记录通道上发送条目）并开始正常执行。如果主 VM 发生故障，则备份 VM 应类似地上线，但该过程稍微复杂一些。由于其执行中的滞后，备份 VM 可能会收到并确认许多日志条目，但由于备份 VM 尚未达到执行中的适当点，因此尚未被使用。备份 VM必须继续从日志条目中重放其执行，直到它使用了最后一个日志条目。此时，备份 VM 将停止重放模式并开始作为普通 VM 执行。本质上，备份 VM 已被提升为主 VM（现在缺少备份 VM）。

由于它不再是备份 VM，新的主 VM
现在将在访客操作系统执行输出操作时将输出生成到外部世界。
在转换到
正常模式期间，可能需要一些特定于设备的操作
才能使此输出正常发生。特别是，
出于网络目的，VMware FT 会自动
在网络上广播新的主 VM 的 MAC 地址，
以便物理网络交换机知道新的主 VM 位于
哪个服务器上。此外，
新提升的主 VM 可能需要重新发布一些
磁盘 IO（如第 3.4 节中所述）。

有很多可能的方法来尝试检测
主 VM 和备份 VM 的故障。VMware FT 使用 UDP
心跳信号在运行容错
VM 的服务器之间进行检测，以检测服务器何时可能崩溃。此外，
VMware FT 监视从
主 VM 发送到备份 VM 的日志流量以及从备份 VM 发送到主 VM 的确认。
由于
定时的定期中断，日志流量应该是定期的，
并且永远不会停止，除非访客操作系统出现故障。因此，日志条目或确认流的中断可能表明
VM 发生故障。如果心跳信号或
日志流量停止的时间超过特定超时时间
（大约几秒钟），则声明发生故障。

但是，任何此类故障检测方法都容易受到
脑裂问题的影响。如果备份服务器停止接收
来自主服务器的心跳信号，则可能表明
主服务器已发生故障，或者可能只是意味着所有
网络连接已在仍在运行的
服务器之间丢失。如果备份 VM 然后上线，而主服务器 VM 实际上仍在运行，则可能会出现数据损坏以及与 VM 通信的客户端出现问题。因此，我们必须确保在检测到故障时，只有主 VM 或备份 VM 中的一个上线。为了避免脑裂问题，我们使用共享存储，该存储存储 VM 的虚拟磁盘。当主 VM或备份 VM 想要上线时，它会在共享存储上执行原子测试和设置操作。如果操作成功，则允许 VM 上线。如果操作失败，则另一个 VM 必须已经上线，因此当前 VM 实际上会停止自身（“自杀”）。如果VM 在尝试执行原子操作时无法访问共享存储，则它只会等到可以访问为止。请注意，如果由于存储网络中的某些故障而无法访问共享存储，则VM 可能无论如何都无法执行有用的工作，因为虚拟磁盘位于同一共享存储上。因此，使用共享存储来解决脑裂情况不会引入任何额外的不可用性。

设计的最后一个方面是，一旦发生故障
并且其中一个 VM 上线，VMware FT 会自动
通过在另一台主机上启动新的备份 VM 来恢复冗余。
虽然大多数
先前的工作中未涵盖此过程，但它是使容错
VM 有用的基础，并且需要仔细设计。更多详细信息在
第 3.1 节中给出。

3. FT 的实际实施

第2节描述了我们的FT的基本设计和协议。然而，要创建一个可用、健壮和自动的系统，还有许多其他组件必须进行设计和实现。

3.1 启动和重启FT虚拟机

必须设计的最大的附加组件之一是启动与主虚拟机处于相同状态的备份虚拟机的机制。当故障发生后重新启动备份虚拟机时，也将使用此机制。因此，此机制必须适用于处于任意状态（即不仅仅是启动）的正在运行的主虚拟机。此外，我们希望该机制不会显着中断主虚拟机的执行，因为这会影响虚拟机的任何当前客户端。

对于VMware FT，我们采用了VMware vSphere现有的VMotion功能。VMware VMotion [10]允许将正在运行的虚拟机从一台服务器迁移到另一台服务器，且中断最小——虚拟机暂停时间通常不到一秒。我们创建了一种修改形式的VMotion，可以在远程服务器上创建虚拟机的精确运行副本，但不会销毁本地服务器上的虚拟机。也就是说，我们修改后的FT VMotion 将虚拟机克隆到远程主机，而不是迁移它。FT VMotion 还设置了一个日志通道，并使源虚拟机进入日志记录模式作为主虚拟机，并使目标虚拟机进入重放模式作为新的备份。与正常的VMotion一样，FT VMotion通常会中断主虚拟机的执行不到一秒。因此，在正在运行的虚拟机上启用FT是一个简单、非破坏性的操作。

启动备份虚拟机的另一个方面是选择在其上运行的服务器。容错虚拟机在可以访问共享存储的服务器集群中运行，因此所有虚拟机通常可以在集群中的任何服务器上运行。这种灵活性允许VMware vSphere即使在以下情况下也能恢复FT冗余

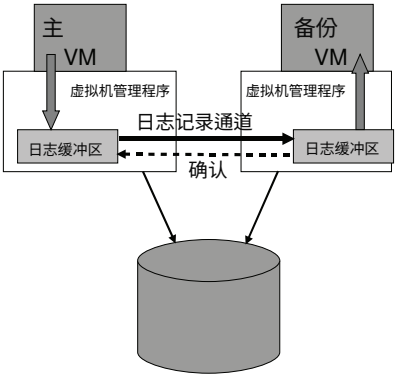

图3：FT日志记录缓冲区和通道。

一个或多个服务器发生故障。VMware vSphere 实现了集群服务，用于维护管理和资源信息。当发生故障并且主 VM 现在需要一个新的备份 VM 来重新建立冗余时，主 VM 会通知集群服务它需要一个新的备份。集群服务根据资源使用情况和其他约束确定运行备份 VM 的最佳服务器，并调用 FT VMotion 来创建新的备份 VM。结果是 VMware FT 通常可以在服务器发生故障后的几分钟内重新建立 VM 冗余，而不会对容错 VM 的执行产生任何明显的中断。

3.2 管理日志通道

在管理日志通道上的流量时，有许多有趣的实现细节。在我们的实现中，hypervisor 为主 VM 和备份 VM 维护一个大型的日志条目缓冲区。当主 VM 执行时，它会将日志条目生成到日志缓冲区中，类似地，备份 VM 从其日志缓冲区中消耗日志条目。主 VM 的日志缓冲区的内容会尽快刷新到日志通道，并且日志条目会尽快从日志通道读取到备份的日志缓冲区中。备份会向主服务器发送确认信息，每次它从网络读取一些日志条目到其日志缓冲区中时。这些确认信息允许 VMware FT 确定何时可以发送被输出规则延迟的输出。图 3 说明了这个过程。

如果备份 VM 在需要读取下一个日志条目时遇到一个空的日志缓冲区，它将停止执行，直到有新的日志条目可用。由于备份 VM 不与外部通信，这种暂停不会影响 VM 的任何客户端。类似地，如果主 VM 在需要写入日志条目时遇到一个满的日志缓冲区，它必须停止执行，直到日志条目可以被刷新出去。这种执行停止是一种自然的流量控制机制，当主 VM 以太快的速率生成日志条目时，它会减慢主 VM 的速度。然而，这种暂停会影响 VM 的客户端，因为主 VM 将完全停止并且无响应，直到它可以记录其条目并继续执行。因此，我们的实现必须被设计为最小化主日志缓冲区填满的可能性。

主日志缓冲区可能填满的一个原因是备份 VM 执行得太慢，因此消耗日志条目也太慢。一般来说，备份 VM 必须能够以与主 VM 记录执行大致相同的速度重放执行。幸运的是，在 VMware 中记录和重放确定性重放的开销大致相同。然而，如果托管备份 VM 的服务器负载过重（因此资源过度分配），则备份 VM 可能无法获得足够的 CPU 和内存资源来以与主 VM 相同的速度执行，尽管备份 hypervisor 的 VM 调度程序尽了最大努力。

除了避免日志缓冲区填满时出现意外暂停之外，我们不希望执行延迟变得太大的另一个原因是。如果主 VM 发生故障，备份 VM 必须通过重放它已经确认的所有日志条目来“赶上”，然后才能上线并开始与外部世界通信。完成重放的时间基本上是故障发生时的执行延迟时间，因此备份上线的时间大致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过长（超过一秒），因为这会大大增加故障切换时间。

因此，我们有一个额外的机制来减慢主 VM 的速度，以防止备份 VM 落后太多。在我们的发送和确认日志条目的协议中，我们发送额外的信息来确定主 VM 和备份 VM 之间的实时执行延迟。通常，执行延迟小于 100 毫秒。如果备份 VM 开始出现明显的执行延迟（例如，超过 1 秒），VMware FT 会通过通知调度程序给它稍微少一点的 CPU（最初只是几个百分点）来开始减慢主 VM 的速度。我们使用一个慢反馈循环，它会尝试逐渐确定主 VM 的适当 CPU 限制，这将允许备份 VM 匹配其执行。如果备份 VM 继续落后，我们将继续逐渐降低主 VM 的 CPU 限制。

相反，如果备份 VM 赶上来，我们会逐渐增加主 VM 的 CPU 限制，直到备份 VM 恢复到略有延迟的状态。

请注意，主 VM 的这种减速非常罕见，通常只在系统承受极端压力时才会发生。第 5 节的所有性能数字都包括任何此类减速的成本。

3.3 FT虚拟机上的操作

另一个实际问题是处理可能应用于主虚拟机的各种控制操作。
例如，如果主虚拟机被显式关闭，那么备份虚拟机也应该被停止，并且不应该尝试上线。再举一个例子，对主机的任何资源管理更改（例如增加CPU份额）也应该应用于备份。对于这些类型的操作，特殊的控制条目通过日志通道从主虚拟机发送到备份虚拟机，以便在备份虚拟机上执行适当的操作。

一般来说，虚拟机上的大多数操作应该只在主虚拟机上启动。然后，VMware FT会发送任何必要的控制条目，以导致备份虚拟机上发生相应的更改。唯一可以在主虚拟机和备份虚拟机上独立完成的操作是VMotion。也就是说，主虚拟机和备份虚拟机可以独立地VMotion到其他主机。请注意，VMware FT确保两个虚拟机都不会移动到另一个虚拟机所在的服务器上，

因为这种情况将不再提供容错能力。

主虚拟机的VMotion比正常的VMotion增加了一些复杂性，因为备份虚拟机必须在适当的时间从源主虚拟机断开连接并重新连接到目标主虚拟机。备份虚拟机的VMotion也有类似的问题，但增加了一个额外的复杂性。

对于正常的VMotion，我们要求在VMotion上的最终切换发生时，所有未完成的磁盘IO都处于静止状态（即已完成）。对于主虚拟机，这种静止可以通过等待物理IO完成并将这些完成传递给虚拟机来轻松处理。但是，对于备份虚拟机，没有简单的方法可以在任何需要的时间点完成所有IO，因为备份虚拟机必须重放主虚拟机的执行并在相同的执行点完成IO。主虚拟机可能正在运行一个工作负载，在该工作负载中，在正常执行期间始终存在正在进行的磁盘IO。VMware FT有一种独特的方法来解决这个问题。当备份虚拟机处于VMotion的最终切换点时，它会通过日志通道请求主虚拟机暂时停止其所有IO。然后，备份虚拟机的IO也会自然地在单个执行点静止，因为它重放主虚拟机执行的静止操作。

3.4 磁盘IO的实现问题

关于磁盘 IO，存在许多微妙的实现问题。首先，鉴于磁盘操作是非阻塞的，因此可以并行执行，访问同一磁盘位置的并发磁盘操作可能导致不确定性。此外，我们的磁盘 IO 实现直接使用 DMA 与虚拟机内存进行数据传输，因此访问相同内存页面的并发磁盘操作也可能导致不确定性。我们的解决方案通常是检测任何此类 IO 竞争（这种情况很少见），并强制此类竞争磁盘操作在主服务器和备份服务器上以相同的方式顺序执行。

其次，磁盘操作也可能与虚拟机中应用程序（或操作系统）的内存访问发生竞争，因为磁盘操作通过 DMA 直接访问虚拟机的内存。
例如，如果虚拟机中的应用程序/操作系统在读取一个内存块，而同时磁盘也在对该块进行读取，则可能会出现不确定的结果。这种情况也不太可能发生，但我们必须检测到它并在发生时处理它。一种解决方案是在磁盘操作的目标页面上临时设置页面保护。如果虚拟机恰好访问了也是未完成磁盘操作目标的页面，页面保护会导致陷阱，并且可以暂停虚拟机直到磁盘操作完成。因为更改页面上的 MMU 保护是一项昂贵的操作，所以我们选择使用反弹缓冲区。反弹缓冲区是一个临时缓冲区，其大小与磁盘操作访问的内存大小相同。磁盘读取操作被修改为将指定的数据读取到反弹缓冲区，并且仅在 IO 完成时才将数据复制到客户机内存。类似地，对于磁盘写入操作，要发送的数据首先被复制到反弹缓冲区，并且磁盘写入被修改为从反弹缓冲区写入数据。反弹缓冲区的使用会降低磁盘操作的速度，但我们没有看到它导致任何明显的性能损失。

第三，当主服务器发生故障并且备份接管时，与主服务器上未完成（即未完成）的磁盘 IO 相关的一些问题。没有办法对于新提升的主虚拟机，无法确定磁盘 IO 是否已发送到磁盘或已成功完成。此外，由于磁盘 IO 未在备份虚拟机上外部发出，因此当新提升的主虚拟机继续运行时，不会有针对它们的显式 IO 完成，这最终会导致虚拟机中的客户机操作系统启动中止或重置过程。我们可以发送一个错误完成，指示每个 IO 都失败了，因为即使 IO 成功完成，返回错误也是可以接受的。但是，客户机操作系统可能对来自其本地磁盘的错误反应不佳。相反，我们在备份虚拟机的上线过程中重新发出挂起的 IO。因为我们已经消除了所有竞争，并且所有 IO 都直接指定了访问哪些内存和磁盘块，所以即使这些磁盘操作已经成功完成，也可以重新发出这些磁盘操作（即它们是幂等的）。

3.5 网络 IO 的实现问题

VMware vSphere 为 VM 网络提供了许多性能优化。其中一些优化基于虚拟机监控程序异步更新虚拟机的网络设备状态。例如，接收缓冲区可以由虚拟机监控程序直接更新，而虚拟机正在执行。不幸的是，这些对虚拟机状态的异步更新增加了不确定性。除非我们能保证所有更新都发生在主服务器和备份服务器上指令流中的同一点，否则备份服务器的执行可能会与主服务器的执行不同。

FT 网络仿真代码的最大变化是禁用了异步网络优化。

异步更新 VM 环的代码
带有传入数据包的缓冲区已被修改为强制客户机陷入虚拟机监控程序，在那里它可以记录更新并将其应用于 VM。类似地，通常异步地从传输队列中提取数据包的代码已针对 FT 禁用，而是通过陷入虚拟机监控程序来完成传输（如下所述除外）。

消除网络设备异步更新，并结合第2.2节中描述的延迟发送数据包，给网络带来了一些性能挑战。我们采取了两种方法来提高运行FT时VM的网络性能。首先，我们实施了集群优化，以减少VM陷阱和中断。当VM以足够的比特率流式传输数据时，虚拟机监控程序可以为每组数据包执行一次传输陷阱，并且在最佳情况下，可以零陷阱，因为它可以在接收新数据包时传输数据包。同样，虚拟机监控程序可以通过仅发布一组数据包的中断来减少VM的传入数据包中断数量。

我们在网络方面的第二个性能优化涉及减少传输数据包的延迟。如前所述，虚拟机监控程序必须延迟所有传输的数据包，直到它从备份收到相应日志条目的确认。减少传输延迟的关键是减少向备份发送日志消息并获得确认所需的时间。我们在这方面的主要优化包括确保发送和接收日志条目和确认都可以在没有任何线程上下文切换的情况下完成。VMware vSphere虚拟机监控程序允许向TCP堆栈注册函数，这些函数将在接收到TCP数据时从延迟执行上下文（类似于Linux中的tasklet）调用。这al-

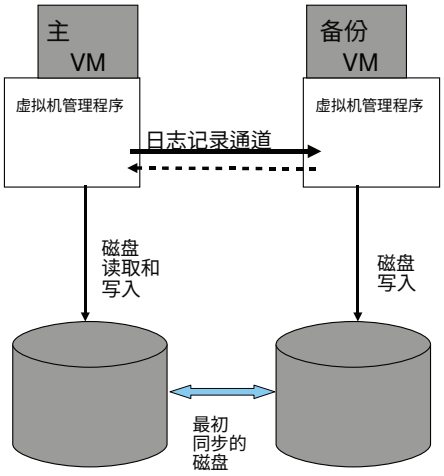

图4：FT非共享磁盘配置。

lows us to quickly handle any incoming log messages on the backup and any acknowledgments received by the primary without any thread context switches. In addition, when the primary VM enqueues a packet to be transmitted, we force an immediate log flush of the associated output log entry (as described in Section 2.2) by scheduling a deferred-execution context to do the flush.

4. 设计替代方案

在我们的VMware FT实现中，我们探索了许多有趣的设计替代方案。在本节中，我们将探讨其中的一些替代方案。

4.1 共享磁盘与非共享磁盘

在我们的默认设计中，主虚拟机和备份虚拟机共享相同的虚拟磁盘。因此，如果发生故障转移，共享磁盘的内容自然是正确的并且可用。
本质上，共享磁盘被认为是主虚拟机和备份虚拟机外部的，因此对共享磁盘的任何写入都被认为是与外部世界的通信。因此，只有主虚拟机才实际写入磁盘，并且对共享磁盘的写入必须根据输出规则进行延迟。

另一种设计是让主虚拟机和备份虚拟机拥有单独的（非共享）虚拟磁盘。在这种设计中，备份虚拟机确实将其所有磁盘写入到其虚拟磁盘，并且这样做，它自然会使其虚拟磁盘的内容与主虚拟机的虚拟磁盘的内容保持同步。图 4 说明了这种配置。在非共享磁盘的情况下，虚拟磁盘本质上被认为是每个虚拟机的内部状态的一部分。因此，主虚拟机的磁盘写入不必根据输出规则进行延迟。在主虚拟机和备份虚拟机无法访问共享存储的情况下，非共享设计非常有用。发生这种情况可能是因为共享存储不可用或太昂贵，或者因为运行主虚拟机和备份虚拟机的服务器相距很远（“长距离 FT”）。非共享设计的一个缺点是，首次启用容错时，必须以某种方式显式同步虚拟磁盘的两个副本。此外，磁盘可能会在发生故障后不同步，因此

在发生故障后重新启动备份虚拟机时，必须显式地重新同步它们。也就是说，FT VMotion 不仅必须同步主虚拟机和备份虚拟机的运行状态，还必须同步它们的磁盘状态。

在非共享磁盘配置中，可能没有共享存储可用于处理脑裂情况。在这种情况下，系统可以使用一些其他的外部仲裁器，例如主服务器和备份服务器都可以与之通信的第三方服务器。如果服务器是具有两个以上节点的集群的一部分，则系统可以替代地使用基于集群成员资格的多数算法。在这种情况下，只有当虚拟机运行在属于包含大多数原始节点的通信子集群的服务器上时，才允许该虚拟机上线。

4.2 在备份虚拟机上执行磁盘读取

在我们的默认设计中，备份虚拟机从不从其虚拟磁盘（无论是共享的还是非共享的）读取数据。由于磁盘读取被认为是输入，因此很自然地通过日志记录通道将磁盘读取的结果发送到备份虚拟机。
另一种设计是让备份虚拟机执行磁盘读取，从而消除磁盘读取数据的日志记录。对于执行大量磁盘读取的工作负载，此方法可以大大减少日志记录通道上的流量。

但是，这种方法有很多微妙之处。它可能会减慢备份虚拟机的执行速度，因为备份虚拟机必须执行所有磁盘读取，并且如果它们在主虚拟机上完成执行时没有物理完成，则必须等待。

此外，必须做一些额外的工作来处理失败的磁盘读取操作。如果主虚拟机的磁盘读取成功，但备份虚拟机的相应磁盘读取失败，则必须重试备份虚拟机的磁盘读取，直到成功为止，因为备份虚拟机必须获得与主虚拟机内存中相同的数据。相反，如果主虚拟机的磁盘读取失败，则必须通过日志记录通道将目标内存的内容发送到备份虚拟机，因为内存的内容将是不确定的，并且不一定通过备份虚拟机的成功磁盘读取来复制。

最后，如果将此磁盘读取替代方案与共享磁盘配置一起使用，则存在一个微妙之处。如果主虚拟机读取特定的磁盘位置，然后很快写入相同的磁盘位置，则必须延迟磁盘写入，直到备份虚拟机执行了第一次磁盘读取。可以检测并正确处理这种依赖关系，但这会增加实现的复杂性。

在第 5.1 节中，我们给出了一些性能结果，表明在备份上执行磁盘读取可能会导致实际应用程序的吞吐量略有降低（1-4%），但也可以显着降低日志记录带宽。因此，在日志记录通道带宽非常有限的情况下，在备份虚拟机上执行磁盘读取可能很有用。

5. 性能评估

在本节中，我们将对 VMware FT 在许多应用程序工作负载和网络基准测试中的性能进行基本评估。对于这些结果，我们在相同的服务器上运行主虚拟机和备份虚拟机，每台服务器都有八个 Intel Xeon 2.8 Ghz CPU 和 8 Gbytes 的 RAM。这些服务器通过 10 Gbit/s 的交叉网络连接，但在所有情况下都可以看到，使用的网络带宽远小于 1 Gbit/s。两台服务器都访问它们共享的虚拟

	性能 (FT / 非 FT)	日志记录 带宽
SPECJbb2005	0.98	1.5 Mbits/sec
内核编译	0.95	3.0 兆比特/秒
Oracle Swingbench	0.99	12兆比特/秒
MS-SQL DVD商店	0.94	18兆比特/秒

表1：基本性能结果

	基础 带宽带宽	FT	日志记录 带宽
接收 (1Gb)	940	604	730
发送 (1Gb)	940	855	42
接收 (10Gb)	940	860	990
传输 (10Gb)	940	935	60

表 2：网络传输和接收到客户端的性能（全部以 Mbit/s 为单位），适用于 1Gb 和 10Gb日志记录通道

来自EMC Clariion的磁盘通过标准
4 Gbit/s光纤通道网络连接。用于驱动
某些工作负载的客户端通过1
Gbit/s网络连接到服务器。

我们在性能结果中评估的应用程序如下。SPECJbb2005是一个行业标准Java应用程序基准测试，它非常占用CPU和内存，并且IO操作非常少。内核编译是一个运行Linux内核编译的工作负载。这个工作负载会进行一些磁盘读取和写入，并且由于创建和销毁许多编译过程，因此非常占用CPU和MMU。Oracle Swingbench是一个工作负载，其中Oracle 11g数据库由Swing-bench OLTP（在线事务处理）工作负载驱动。这个工作负载会进行大量的磁盘和网络IO，并且有八十个并发数据库会话。MS-SQL DVD商店是一个工作负载，其中Microsoft SQL Server 2005数据库由DVD商店基准测试驱动，该基准测试具有十六个并发客户端。

5.1 基本性能结果

表1给出了基本性能结果。对于列出的每个应用程序，第二列给出了在运行服务器工作负载的VM上启用FT时应用程序的性能与在同一VM上未启用FT时的性能之比。性能比率的计算方式是，小于1的值表示FT工作负载较慢。显然，在这些代表性的工作负载上启用FT的开销小于10%。SPECJbb2005完全受计算限制，没有空闲时间，但由于除了计时器中断之外，它具有最少的非确定性事件，因此性能良好。其他工作负载进行磁盘IO并具有一些空闲时间，因此FT的一些开销可能会被FTVM具有较少空闲时间这一事实所掩盖。但是，一般的结论是VMware FT能够以相当低的性能开销支持容错VM。

在该表的第三列中，我们给出了运行这些应用程序时在日志记录通道上发送的数据的平均带宽。对于这些应用程序，日志记录带宽非常合理，并且很容易满足1 Gbit/s网络的需求。实际上，低带宽要求表明多个FT工作负载可以共享同一个1 Gbit/s网络，而不会产生任何负面的性能影响。

对于运行常见客户机操作系统的虚拟机，如 Linux 和 Windows，我们发现当客户机操作系统空闲时，典型的日志记录带宽为 0.5-1.5 Mbits/sec。
“空闲”带宽主要是记录定时器中断传递的结果。对于具有活动工作负载的虚拟机，日志记录带宽主要由必须发送到备份的网络和磁盘输入决定——接收到的网络数据包和从磁盘读取的磁盘块。因此，日志记录带宽可能要大得多高于表 1 中测量的那些，适用于具有非常高的网络接收或磁盘读取带宽的应用程序。对于这些类型的应用程序，日志记录通道的带宽可能成为瓶颈，特别是当日志记录通道还有其他用途时。

对于许多实际应用程序来说，日志记录通道上所需的相对较低的带宽使得基于重放的容错对于使用非共享磁盘的远距离配置非常有吸引力。对于主服务器和备份服务器可能相隔 1-100 公里的远距离配置，光纤可以轻松支持 100-1000 Mbit/s 的带宽，延迟小于 10 毫秒。

对于表 1 中的应用程序，100-1000 Mbit/s 的带宽应该足以获得良好的性能。但是请注意，主服务器和备份服务器之间的额外往返延迟可能会导致网络和磁盘输出延迟最多 20 毫秒。远距离配置仅适用于客户端可以容忍每个请求的这种额外延迟的应用程序。

对于两个磁盘密集型最高的应用程序，我们测量了在备份 VM 上执行磁盘读取（如第 4.2 节中所述）与通过日志记录通道发送磁盘读取数据对性能的影响。对于 Oracle Swingbench，在备份 VM 上执行磁盘读取时，吞吐量降低约 4%；对于 MS-SQL DVD Store，吞吐量降低约 1%。同时，对于 Oracle Swingbench，日志记录带宽从 12 Mbits/sec 降低到 3 Mbits/sec，对于 MS-SQL DVD Store，日志记录带宽从 18 Mbits/sec 降低到 8 Mbits/sec。显然，对于具有更大磁盘读取带宽的应用程序，带宽节省可能会更大。正如第 4.2 节中提到的，预计在备份 VM 上执行磁盘读取时，性能可能会稍差一些。但是，对于日志记录通道带宽受限的情况（例如，远距离配置），在备份 VM 上执行磁盘读取可能很有用。

5.2 网络基准测试

由于多种原因，网络基准测试对我们的系统来说可能非常具有挑战性。首先，高速网络可能具有非常高的中断率，这需要以非常高的速率记录和重放异步事件。其次，以高速率接收数据包的基准测试将导致高流量的日志记录，因为所有此类数据包都必须通过日志记录通道发送到备份。

第三，发送数据包的基准测试将受到输出规则的约束，该规则会延迟网络数据包的发送，直到收到来自备份的相应确认。这种延迟会增加测得的到客户端的延迟。这种延迟也可能降低到客户端的网络带宽，因为网络协议（如 TCP）可能

随着往返延迟的增加，降低网络传输速率。

表 2 给出了我们使用标准 netperf 基准测试进行的一些测量的结果。在所有这些测量中，客户端 VM 和主 VM 通过 1 Gbit/s 的网络连接。前两行给出了当主服务器和备份主机通过 1 Gbit/s 的日志记录通道连接时的发送和接收性能。第三行和第四行给出了当主服务器和备份服务器通过 10 Gbit/s 的日志记录通道连接时的发送和接收性能，该通道不仅具有更高的带宽，而且还具有比 1 Gbit/s 网络更低的延迟。粗略地说，1 Gbit/s 连接的虚拟机监控程序之间的 ping 时间约为 150 微秒，而 10 Gbit/s 连接的 ping 时间约为 90 微秒。

当未启用 FT 时，主 VM 可以实现接近（940 Mbit/s）1 Gbit/s 线路速率的发送和接收。当为接收工作负载启用 FT 时，日志记录带宽非常大，因为所有传入的网络数据包都必须在日志记录通道上发送。因此，日志记录通道可能会成为瓶颈，如 1 Gbit/s 日志记录网络的结果所示。对于 10 Gbit/s 日志记录网络，效果要小得多。当为传输工作负载启用 FT 时，不会记录已传输数据包的数据，但仍必须记录网络中断。

日志记录带宽要低得多，因此可实现的网络传输带宽高于网络接收带宽。总的来说，我们看到 FT 可以在非常高的传输和接收速率下显着限制网络带宽，但仍然可以实现较高的绝对速率。

6. 相关工作

Bressoud 和 Schneider [3] 描述了通过完全包含在虚拟机监控程序级别的软件为虚拟机实现容错的最初想法。他们证明了通过 HP PA-RISC 处理器的服务器原型，使备份虚拟机与主虚拟机保持同步的可行性。但是，由于 PA-RISC 体系结构的限制，他们无法实现完全安全、隔离的虚拟机。此外，他们没有实现任何故障检测方法，也没有尝试解决第 3 节中描述的任何实际问题。

更重要的是，他们对其 FT 协议施加了许多不必要的约束。首先，他们强加了 epoch 的概念，其中异步事件被延迟到一组间隔的末尾。epoch 的概念是不必要的——他们可能强加了它，因为他们无法足够有效地重放单个异步事件。其次，他们要求主 VM基本上停止执行，直到备份收到并确认所有先前的日志条目。但是，只有输出本身（例如网络数据包）必须延迟——主 VM 本身可以继续执行。

Bressoud [4] 描述了一个在操作系统 (Unixware) 中实现容错的系统，因此为在该操作系统上运行的所有应用程序提供容错功能。系统调用接口成为必须确定性复制的操作集。

这项工作与基于虚拟机管理程序的工作有相似的局限性和设计选择。

Napper等人[9]以及Friedman和Kama[7]描述了容错Java虚拟机的实现。它们在发送有关以下信息方面采用了与我们类似的设计在日志通道上输入和非确定性操作。与Bressoud一样，他们似乎没有专注于检测故障并在故障后重建容错能力。此外，他们的实现仅限于为在Java虚拟机中运行的应用程序提供容错能力。

这些系统试图处理多线程Java应用程序的问题，但要求所有数据都受到锁的正确保护，或者强制对共享内存的访问进行序列化。

Dunlap等人[6]描述了一种确定性重放的实现，该实现旨在调试半虚拟化系统上的应用程序软件。我们的工作支持在虚拟机内部运行的任意操作系统，并为这些虚拟机实现容错支持，这需要更高水平的稳定性和性能。

Cully等人[5]描述了一种支持容错虚拟机及其在名为Remus的项目中的实现的替代方法。通过这种方法，主虚拟机的状态在执行期间被重复检查点，并发送到备份服务器，该服务器收集检查点信息。检查点必须非常频繁地执行（每秒多次），因为外部输出必须延迟到发送并确认后续检查点为止。这种方法的优点是它同样适用于单处理器和多处理器虚拟机。

主要问题是，这种方法需要非常高的网络带宽，才能将增量更改发送到每个检查点的内存状态。在[5]中给出的Remus的结果显示，当尝试使用1 Gbit/s网络连接每秒进行40次检查点以传输内存状态的更改时，内核编译和SPECweb基准测试的减速为100%到225%。有许多优化可能有助于降低所需的网络带宽，但尚不清楚使用1 Gbit/s连接是否可以实现合理的性能。相比之下，我们基于确定性重放的方法可以实现低于10%的开销，并且对于几个实际应用程序，主服务器和备份主机之间所需的带宽小于20 Mbit/s。

7. 结论与未来工作

我们已经在VMware vSphere中设计并实现了一个高效且完整的系统，该系统为在集群中的服务器上运行的虚拟机提供容错（FT）。我们的设计基于通过使用VMware确定性重放在另一台主机上的备份虚拟机复制主虚拟机的执行。如果运行主虚拟机的服务器发生故障，则备份虚拟机将立即接管，而不会中断或丢失数据。

总的来说，在商品硬件上，VMware FT下的容错虚拟机的性能非常出色，并且对于某些典型应用程序，开销低于10%。VMware FT的大部分性能成本来自使用VMware确定性重放来保持主虚拟机和备份虚拟机同步的开销。因此，VMware FT的低开销源于VMware确定性重放的效率。此外，保持主虚拟机和备份虚拟机同步所需的日志记录带宽通常非常小，通常小于20 Mbit/s。由于在大多数情况下，日志记录带宽非常小，因此似乎可以实现主虚拟机和备份虚拟机相隔很远距离（1-100公里）的配置。因此，VMware FT可以用于以下场景：

还可以防止整个站点发生故障的灾难。值得注意的是，日志流通常具有很高的可压缩性，并且简单的压缩技术可以通过少量的额外 CPU 开销来显着降低日志记录带宽。

我们使用 VMware FT 获得的结果表明，容错虚拟机的有效实现可以建立在确定性重放的基础上。这样的系统可以透明地为运行任何操作系统和应用程序的虚拟机提供容错能力，并且开销最小。但是，对于客户而言，容错虚拟机系统要有用，它还必须是健壮的、易于使用的和高度自动化的。一个可用的系统需要许多其他组件，而不仅仅是虚拟机的复制执行。特别是，VMware FT 在发生故障后会自动恢复冗余，方法是在本地集群中找到合适的服务器并在该服务器上创建新的备份虚拟机。通过解决所有必要的问题，我们已经演示了一个可用于客户数据中心中的实际应用程序的系统。

通过确定性重放实现容错的权衡之一是，目前确定性重放仅针对单处理器虚拟机有效实现。

但是，单处理器虚拟机对于各种工作负载来说已经足够了，尤其是在物理处理器不断变得更强大的情况下。此外，许多工作负载可以通过使用许多单处理器虚拟机来扩展，而不是通过使用一个更大的多处理器虚拟机来扩展。多处理器虚拟机的高性能重放是一个活跃的研究领域，并且可以通过微处理器中的一些额外硬件支持来实现。一个有趣的方向可能是扩展事务内存模型以促进多处理器重放。

将来，我们也有兴趣扩展我们的系统以处理部分硬件故障。通过部分硬件故障，我们指的是服务器中功能或冗余的部分丢失，这不会导致数据损坏或丢失。一个例子是与虚拟机的全部网络连接丢失，或者物理服务器中冗余电源的丢失。如果在运行主虚拟机的服务器上发生部分硬件故障，在许多情况下（但并非全部），立即故障转移到备份虚拟机将是有利的。这样的故障转移可以立即为关键虚拟机恢复完整服务，并确保虚拟机快速从潜在不可靠的服务器上移开。

致谢

我们要感谢 Krishna Raja，他生成了许多性能结果。有很多人参与了 VMware FT 的实施。确定性重放的核心实施者（包括对各种虚拟设备的支持）和基本 FT 功能包括 Lan Huang、Eric Lowe、Slava Malyugin、Alex Mirgorodskiy、Kaustubh Patil、Boris Weissman、Petr Van-drovec 和 Min Xu。此外，还有许多其他人参与了 VMware vCenter 中 FT 的更高级别管理。Karyn Ritter 在管理大部分工作方面做得非常出色。

8. 参考文献

[1] Alsberg, P., and Day, J.分布式资源弹性共享原则。见软件工程第二次国际会议论文集(1976)，第 627–644 页。

[2]AMD公司.AMD64架构程序员手册。加利福尼亚州森尼维尔。

[3]布雷苏德，T.，和施耐德，F.基于虚拟机监控程序的容错。在SOSP 15会议论文集（12月）中。1995).

[4]布雷苏德，T. C.TFT：一种用于应用程序透明容错的软件系统。在第二十八届年度容错计算国际研讨会论文集中（1998年6月），第128-137页。

[5]卡利，B.，勒菲弗，G.，迈耶，D.，费利，M.，哈奇森，N.，和沃菲尔德，A.雷姆斯：通过异步虚拟机复制实现高可用性。在第五届USENIX网络系统设计与实现研讨会论文集中（2008年4月），第161-174页。

[6]邓拉普，G. W.，金，S. T.，西纳，S.，巴斯拉伊，M.，和陈，P. M.ReVirt：通过虚拟机日志记录和重放实现入侵分析。在2002年操作系统设计与实现研讨会论文集中（12月）。2002).

[7]弗里德曼，R.，和卡玛，A.透明容错Java虚拟机。在可靠分布式系统会议论文集中（2003年10月），第319-328页。

[8]英特尔公司.英特尔 ^AR◯64和IA-32架构软件开发人员手册。加利福尼亚州圣克拉拉。

[9]纳珀，J.，阿尔维西，L.，和文，H.一种容错Java虚拟机。在国际可靠系统与网络会议论文集中（2002年6月），第425-434页。

[10]尼尔森，M.，林，B.-H.，和哈钦斯，G.虚拟机的快速透明迁移。在2005年USENIX年度技术会议论文集中（2005年4月）。

[11] Nightingale, E. B., Veeraraghavan, K., Chen, P. M., 和 Flinn, J. 重新思考同步。在 2006 年操作系统设计与实现研讨会论文集（2006 年 11 月）。

[12] Schlicting, R., 和 Schneider, F. B.Fail-stop 处理器：一种设计容错计算系统的方法。 ACM 计算调查 1, 3（1983 年 8 月），222–238。

[13]Schneider, F. B. 使用状态机方法实现容错服务：教程。ACM 计算调查 22, 4 (1990 年 12 月), 299–319。

[14]Stratus Technologies. 受益于 Stratus 持续处理技术：Microsoft Windows Server 环境的自动99.999% 正常运行时间。网址为 http://www.stratus.com/~/media/-Stratus/Files/Resources/WhitePapers/continuous-processing-for-windows.pdf，六月2009.

[15] Xu, M., Malyugin, V., Sheldon, J., Venkitachalam, G., 和 Weissman, B. ReTrace: 使用虚拟机收集执行跟踪确定性重放。在 2007 年建模、基准测试和模拟研讨会论文集（2007 年 6 月）。