容错虚拟机分布式系统的设计

摘 要

我们已经实现了一个提供容错虚拟机的商业级别的企业级系统,它是基于通过另一台服务器上的备份虚拟机复制运行的主虚拟机（VM）的方法。我们在 VMware vSphere 4.0 中设计了一个完整的系统，该系统易于使用，可在商品服务器上运行，并且通常 会使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，使主虚拟机和辅助虚拟机保持同步执行所需的数据带宽小于 20 Mbit/s，这允许在更长的距离上实现容错能力。一个易于使用的商用系统，在故障后自动恢复冗余，除了执行复制的 VM 之外，还需要许多其他组件。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VM 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许多实现细节,并提 供了微基准测试(micro-benchmarks)和实际应用的性能结果。

1. 介绍

实现容错服务器的常用方法是primary/backup(主/备用)方法[1]，如果主服务器发生故障，备用服务器将始终可以接管。备用服务器的状态必须始终保持与主服务器几乎相同，以便备用服务器可以在主服务器发生故障时立即接管，并且这种故障对外部客户端隐藏并且没有数据丢失。在备份服务器上复制状态的一种方法是将对主数据库的所有状态（包括 CPU，内存和 I / O 设备）的更改几乎连续不断地传送到备份服务器上。但是，发送此状态所需的带宽（特别是内存更改）可能非常大。

使用带宽少得多的另一种复制服务器的方法称为state-machine方法[13]。想法是将服务器建模为确定性的状态机(deterministic state machine)，该方法通过从相同的初始状态启动服务器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的某些操作不确定，因此必须使用额外的协调以确保主服务器和备份保持同步。但是，保持主数据库和备份同步所需的额外信息量远远少于主数据库中正在更改的状态（主要是内存更新）量。

实现协调以确保确定性的物理服务器的执行[14]是困难的， 尤其是随着处理器频率的提高。相反，在虚拟机管理程序之上运行的虚拟机（VM）是用于实现state-machine方法的出色平台。可以将 VM 视为定义良好(well-defined)的state-machine。其操作是要虚拟化的计算机(包括其所有设备)的操作。与物理服务器一样，VM 具有一些不确定性的操作（例如，读取一天中的时钟或中断的传递），因此必须向备份发送额外的信息以确保其保持同步。由于管理程序对 VM 的执行具有完全控制权，包括所有输入的传递，因此管理程序能够捕获有关primary VM 上非确定性操作的所有必要信息，并在备份 VM 上正确地重放这些操作。

因此，无需对硬件进行任何修改，就可以在商用硬件上为虚拟机实现state-machine方法，从而可以为最新的微处理器立即实现容错功能。另外，state-machine方法所需的低带宽允许更大程度地分离主数据库和备用数据库。例如，复制的虚拟机可以在园区内分布的物理机上运行，这比在同一建筑物中运行的 VM 提供更高的可靠性。

我们已经在 VMware vSphere 4.0 平台上使用主/备份方法实现了容错 VM，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 实施了完整的 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能力。允许我们记录主数据库的执行并确保备份执行相

同的基础技术称为确定性重放(determinstic replay) [15] 。 VMware vSphere Fault Tolerance（FT）基于确定性重放，但添加了必要的额外协议和功能以构建完整的容错系统。除了提供硬件容错能力外， 我们的系统还可以通过在本地群集中任何可用服务器上启动新的备份虚拟机来自动恢复故障后的冗余。目前，确定性重放和 VMware FT 的生产版本仅支持单处理器 VM。记录和重放多处理器 VM 的执行仍在进行中，存在重大的性能问题，因为对共享内存的每次访问都可能是不确定的操作。

Bressoud 和 Schneider [3]描述了一个基于HP PA-RISC 平台的容错 VM 的原型实现。我们的方法相似，但是出于性能原因，我们进行了一些根本性更改，并研究了许多设计方案来替代。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而言是高效且可用的。与讨论的大多数实际系统类似，我们仅尝试处理fail-stop failures[12]，这是在出现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检测服务器故障的服务。

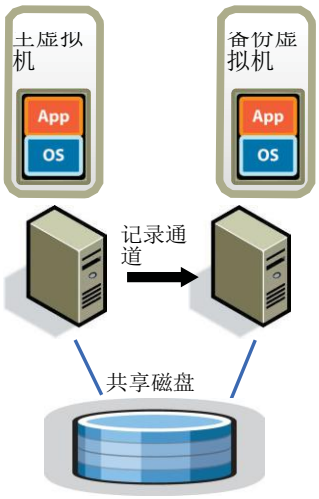


图 1：基本 FT 配置

本文的其余部分安排如下。首先，我们描述基本设计并详细介绍基本协议，以确保在主 VM 发生故障后，如果备份 VM 接管，则不会丢失任何数据。然后，我们详细描述了构建健壮，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 的几种设计选择，并讨论了这些选择之间的权衡。接下来，我们给出一些基准测试和某些实际企业应用程序实施的性能结果。最后，我们描述相关工作并得出结论。

2. BASIC

图 1 显示了我们用于容错 VM 的系统的基本设置。对于一个VM(主 VM),假如我们希望为其提供容错能力，我们会在另一台物理服务器上运行备份 VM，备份VM与主VM几乎同时并且相同地执行命令(尽管时滞很小)以保持同步。我们说这两个虚拟机处于虚拟同步状态(virtual lock-step)。VM 的虚拟磁盘位于共享存储（例如光纤通道或iSCSI 磁盘阵列）上，因此主 VM 和备份 VM 可以访问它们的输入、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通告其存在，因此所有网络输入都将传给主虚拟机。同样，所有其他输入（例如键盘和鼠标）仅进入主 VM。

主 VM 收到的所有输入都将通过称为日志记录通道(logging channel)的网络发送到备份 VM。对于服务器工作负载，主要的输入流量是网络和磁盘。如下文第 2.1 节所述，将根据需要传输其他信息，以确保备份 VM 以与主 VM 相同的方式执行非确定性操作。结果是，备份 VM 始终与主要 VM 执行相同的操作。但是，备份虚拟机的输出由虚拟机管理程序删除，因此只有主虚拟机会产生实际输出，然后将其返回给客户端。如第 2.2 节所述，主虚拟机和备份 VM 遵循特定的协议，包括备用 VM 的明确确认，以确保在主虚拟机发生故障时不会丢失任何数据。

为了检测主虚拟机或备用虚拟机是否发生故障，我们的系统将相关服务器之间的心跳信号与日志记录通道的流量监视结合起来使用。此外，即使存在主服务器和备份服务器之间失去通信的裂脑情况，我们也必须确保仅主虚拟机或备份虚拟机中的一个接管执行。

在以下各节中，我们将提供有关几个重要领域的更多详细信息。在第 2.1 节中，我们将提供有关确定性重播技术的一些详细信息，该技术可确保通过记录通道发送的信息使主虚拟机和备用虚拟机保持同步。在第 2.2 节中，我们描述了 FT 协议的基本规则，该规则可确保在主协议失败时不会丢失任何数据。在第 2.3 节中，我们描述了以正确的方式检测和响应故障的方法。

2.1 确定性回放实现

正如我们已经提到的，备份服务器(或 VM)的执行可以建模为确定性state machine的拷贝。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了完全相同的输入，则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集，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(例如虚拟中断)和非确定性操作(例如读取处理器的时钟周期计数器)也会影响 VM 的状态。这对于运行在任意操作系统和工作负载上的任意VM的拷贝执行提出了三个挑战：(1)正确捕获所有输入和不确定性以确保确定性备份虚拟机所需的执行;(2)正确地将输入和不确定性应用于备份虚拟机;(3)并且确保这样做的方式不会降低性能。此外，x86 微处理器中的许多复杂操作都有不确定的副作用，因此不确定(hence non-deterministic)。捕获这些未定义的副作用并回放它们以产生相同的状态又成为了另一个挑战。

VMware 确定性回放[15]为 VMware vSphere 平台上的 x86 虚拟机完全提供了此功能。确定性回放在写入日志文件的日志条目流中记录 VM 的输入以及与 VM 执行相关的所有可能的不确定性。VM执行可以通过从文件中读取日志条目准确的回放。对于非确定性操作，将记录足够的信息以允许以相同的状态更改和输出来再现操作。对于不确定事件，例如计时器或 IO 完全中断时，也会记录事件发生的确切指令。在回放期间，事件在指令流中的同一point传递。VMware 确定性回放实现了一种有效的事件记录和事件传递机制，该机制采用了多种技术，包括使用与 AMD [2]和 Intel [8]共同开发的硬件性能计数器。

Bressoud 和 Schneider [3]提到将 VM 的执行划分为多个时期，其中诸如中断之类的不确定事件仅在一个时期的末尾传递。epoch 的概念似乎被用作批处理机制，因为将每个中断按照发生的确切指令分别交付是太昂贵了。但是，我们的事件传递机制足够有效，以至于 VMware 确定性回放无需使用epoch。每次中断都会记录下来，并在回放时按适当的指令有效的发送。

2.2 FT Protocol

对于 VMware FT，我们使用确定性回放来生成必要的日志条目以记录主 VM 的执行情况，但是,但是我们没有直接将日志条目(log entries)直接写入硬盘,而是将日志条目通过logging channel发送给备份VM。备份 VM 实时回放日志条目，因此与主 VM 的执行顺序是相同的。然而,我们必须在日志记录通道上使用严格的 FT 协议来扩充日志记录条目，以确保实现容错能力。我们的基本要求如下：

输出要求：如果备份 VM 在主节点发生故障后接管，则备份 VM 将以与主 VM 已发送到外部的所有输出完全一致的方式继续执行。

请注意，在发生故障转移之后（即，备份虚拟机在主虚拟机故障后接管），由于许多不确定性事件，备份虚拟机可能开始执行的方式与主虚拟机将继续执行的方式大不相同。在执行期间发生。但是，只要备份 VM 满足输出要求，在故障转移到备份 VM 的过程中就不会丢失任何外部可见状态或数据，并且客户端将不会发现其服务中断或不一致。

为了确保输出的要求可以通过延迟所有的外部输出(通常指网络数据包)直到备份 VM 接收到所有信息为止，这将使其至少在该输出操作之前可以回放执行。一个必要条件是，备份 VM 必须已经接收在输出操作之前生成的所有日志条目。这些日志条目将使其执行到最后一个日志条目为止(也就是所有的日志操作都要执行完)。但是，假设在主数据库执行输出操作后立即发生故障。备份虚拟机必须知道它必须一直回放直到输出操作为止，并且此时只能“上线(go live)”（停止回放并接管为主虚拟机，如 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处生效，则某些不确定事件（例如，传递给 VM 的计时器中断）可能会在执行输出操作之前更改其执行路径。

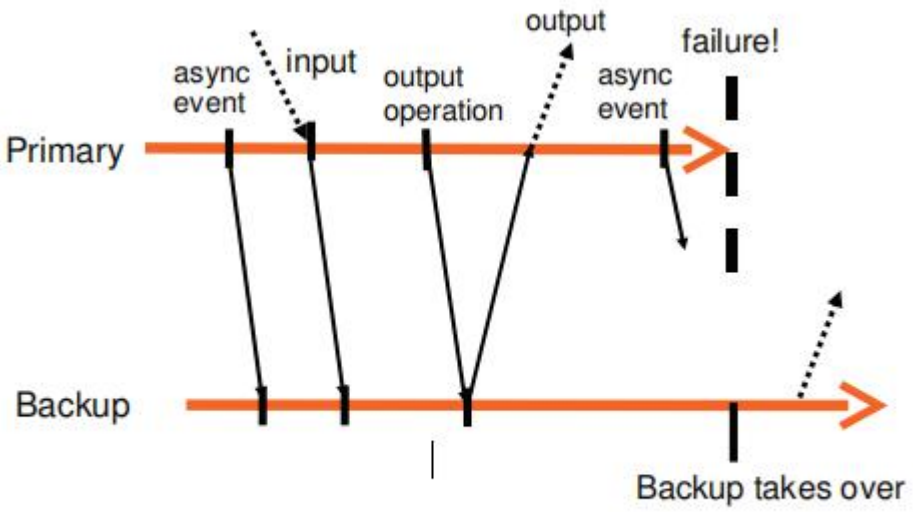


Figure 2: FT Protocol.

鉴于上述限制，执行输出要求的最简单方法是在每个输出操作位置创建特殊的日志条目。然后，可以通过以下特定规则强制执行“输出要求”：

输出规则：在备份 VM 收到并确认与产生输出的操作相关的日志条目之前，主 VM 可能不会将输出发送到外部环境。

如果备份虚拟机已收到所有日志条目，包括用于输出生成操作的日志条目，则备份VM将能够准确地再现该输出点上主VM的状态，因此如果主VM死了，备份VM将正确地达到与该输出一致的状态。相反，如果备份VM接管而未收到所有必要的日志条目，则其状态可能会迅速偏离，从而使其与主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 可以实际进行缓冲， 只要在下一次外部通信之前将其实际写入磁盘即可。

请注意，输出规则并未说明停止主 VM 的执行。我们只需要延迟输出的发送，但是 VM 本身可以继续执行。由于操作系统使用异步中断来进行非阻塞网络和磁盘输出以标志完成，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。相反，先前的工作[3，9]通常表明在完成输出之前必须完全停止主 VM，直到备份 VM 确认了来自主 VM 的所有必要信息。

例如，我们在图 2 中显示了说明 FT 协议要求的图表。该图显示了主 VM 和备用 VM 上事件的时间表。从主行到备用行的箭头表示日志条目的传输，从备用行到主行的箭头表示确认。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，到外部环境的输出被延迟要等到主VM从备份VM接收到它已经接收到与输出操作相关联的日志条目的确认。如果遵循输出规则，则备份虚拟机将 能够以与主数据库的最后输出一致的状态进行接管。

我们不能保证在容错(failover)的情况下所有输出都只产生一次。 如果在主VM要发送输出时不使用具有两阶段提交的事务， 则备份VM无法确定主VM在紧接其最后一个输出之前或之后是否崩溃。幸运的是，网络基础结构(包括 TCP 的常用用法)旨在处理丢失的数据包和相同(重复)的数据包。请注意,在主VM发生故障期间，到主VM的传入数据包也可能会丢失，因此不会传递给备份VM。但是，由于多种与服务器故障无关的原因，传入的数据包也可能会被丢弃，因此编写网络基础结构，操作系统和应用程序都是为了确保它们可以补偿丢失的数据包。

2.3 Detecting and Responding to Failure

如上所述，如果另一个 VM 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 将处于上线状态(go love),即，退出记录模式（并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，则备用VM也应同样上线，但过程要复杂一些。由于备份 VM 的执行滞后，它可能会有许多已接收并应答的日志条目尚未使用， 因为备份 VM 尚未达到执行的适当执行点。备份 VM 必须继续从日志条目回放其执行，直到消耗完最后一个日志条目为止。此时，备份 VM 将停止回放模式并开始以普通 VM 的身份执行。本质上，备份VM已升级为主VM（现在缺少备份VM）。由于它不再是备份 VM，因此，当guest OS 进行输出操作时，新的主 VM 将新产生输出到外部环境。在过渡到普通模式期间，可能需要一些特殊设备的操作才能使此输出正确发生。尤其是，出于联网目的，VMware FT 会自动在网络上通告新的主 VM 的 MAC 地址，以便物理网络交换机将知道新的主 VM 位于哪台服务器上。另外，新提升的主 VM 可能需要重新发行(重新执行输入输出?)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虚拟机和备用虚拟机的故障。VMware FT 在运行容错 VM 的服务器之间使用 UDP 心跳检测来监控服务器何时崩溃。此外，VMware FT 监视从主VM发送到备份 VM 的日志记录流量以及从备份 VM 到主虚拟机发送的应答信息。由于定期的计时器中断，日志流

量应该是正常服务，并且对于活动状态下的OS的用户访问不应让其察觉到中断。因此，日志条目或应答流的停止可能表示 VM 发生故障。如果心跳或日志记录通信的停止时间超过指定的超时时间（大约几秒钟），则声明为故障。

但是，任何这种故障检测方法都容易出现裂脑问题。如果备份服务器停止从主服务器接收心跳信号，则可能表明主服务器发生故障，或者仅表示仍在运行的服务器之间所有网络连接已丢失。此时主VM其实仍在运行,如果这时备份VM上线,可能会出现数据损坏以及客户端与虚拟机通信问题。因此，我们必须确保在检测到故障时，只有一个主虚拟机或备份虚拟机可以运行。为了避免出现脑裂问题，我们使用了共享存储用于存储VM的虚拟磁盘。当主虚拟机或备用虚拟机想要上线时，它将在共享存储上执行一个atomic test-and-set。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 实际上会自行停止(“自杀”)。如果虚拟机在尝试执行原子操作时无法访问共享存储，则它会一直等到可以。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做有用的工作，因为虚拟磁盘位于同一共享存储上。因此，使用共享存储解决裂脑情况不会引入任何额外的不可用性。

该设计的最后一个方面是，一旦发生故障并且其中一个 VM 已上线(是指备份VM上线)，VMware FT 会通过在另一台主机上启动新的备份 VM 来自动恢复冗余。尽管此过程在以前的大多数工作中都没有涉及，但这对于使容错 VM 有用是至关重要的，并且需要仔细设计。更多细节在第 3.1 节中给出。

3. Practical implementation of FT

第 2 节介绍了 FT 的基本设计和协议。但是，要创建一个可用的，健壮的和自动的系统，还必须设计和实现许多其他组件。

3.1 Starting and Restarting FT VMs

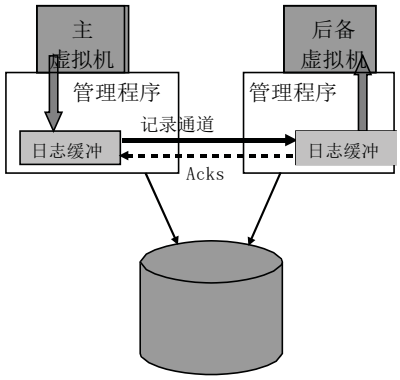
启动备份VM能过渡到与主VM相同状态的机制是必须要设计的额外的最大的组件之一。发生故障后重新启动备份 VM 时，也会使用此机制。因此，该机制必须可用于处于任意状态(即，不仅正在启动)的正在运行的主 VM。此外，我们希望该机制不会显著中断主 VM 的执行，因为这会影响连接VM 的客户端。

对于 VMware FT，我们调整了 VMware vSphere 的现有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，并且中断时间最少，虚拟机暂停时间通常不到一秒钟。我们创建了 VMotion 的修改形式，可以在远程服务器上创建虚拟机的精确运行副本，但不会破坏本地服务器上的 VM。也就是说，我们修改后的 FT VMotion 将 VM 复制到远程主机，而不是迁移它。FT VMotion 还设置了一个日志记录通道，并导致源 VM 作为主数据库进入日志记录模式，而目标 VM 作为新备份进入回放模式。像普通的 VMotion 一样，FT VMotion 通常将主 VM 的执行中断不到一秒钟。因此，在运行中的 VM 上启用 FT 是一项简单，无中断的操作。

启动备份 VM 的另一方面是选择一个可以运行该VM的服务器。容错 VM 在可以访问共享存储

的服务器群集中运行，因此所有 VM 通常可以在群集中的任何服务器上运行。这种灵活性使 VMware vSphere 即使在一台或多台服务器发生故障的情况下也可以还原 FT 冗余。

VMware vSphere 实现了一个群集服务，用于维护管理和资源信息。当发生故障并且主虚拟机立即需要新的备份虚拟机来重新建立冗余时，主虚拟机会通知集群服务它需要一个新的备份。群集服务根据资源使用情况和其他约束条件确定运行备份 VM 的最佳服务器，并调用 FT VMotion 创建新的备份 VM。结果是，VMware FT 通常可以在服务器故障后的几分钟内重新建立 VM 冗余，而所有这些操作都不会在容错 VM 的执行中引起任何明显的中断。



3.2 Managing the Logging Channel

在日志记录通道上管理流量时，有许多有趣的实现细节。 在我们的实现中，虚拟机监控程序维护着一个很大的缓冲区， 用于记录主虚拟机和备用虚拟机的日志信息。当主 VM 执行时，它将在日志缓冲区中生成日志条目，并且类似地，备份 VM 从其日志缓冲区中使用日志条目。主数据库日志缓冲区的内容会尽快刷新到日志记录通道，一旦日志条目到达，日志条目就会从日志记录通道读入备份的日志缓冲区。备份每次从网络 将一些日志条目读取到其日志缓冲区时，备份都会将应答消息发送回主VM。这些应答消息(acks)使 VMware FT 可以确定何时可以发送由“输出规则”延迟的输出。上图说明了此过程。

如果备份虚拟机在需要读取下一个日志条目时遇到空的日志缓冲区，它将停止执行，直到有新的日志条目可用为止。由于备份虚拟机不在外部进行通信，因此该暂停不会影响虚拟机的任何客户端。同样，如果主虚拟机在需要写入日志条目时遇到了完整的日志缓冲区，则它必须停止执行，直到日志条目被flushed out为止。在执行中这种停止是一种自然的流控制机制，当主 VM 生成日志条目的速度过快时，它会降低主 VM 的速度。但是，此暂停会影响 VM 的客户端，因为主 VM 将完全停止且无响应，直到它可以记录其条目并继续执行。因此，我们的实现必须设计为最大程度地减少主日志缓冲区被填满的可能性。

主日志缓冲区可能已满的原因之一是因为备份 VM 执行太慢，因此消耗日志条目太慢。通常，备份 VM必须能够以与主 VM 记录执行大致相同的速度回复执行。幸运的是，VMware 确定性回放中记录和回放的开销大致相同。但是，如果托管备份虚拟机的服务器负载过重,例如负载了其他 VM(因此过度占用资源)，尽管备份虚拟机管理程序的虚拟机调度程序尽了最大努力，但备份虚拟机可能无法获得足够的CPU和内存资源来像主虚拟机一样快速执行。

除了避免日志缓冲区已满时出现意外暂停之外，还有另一个我们为什么不希望执行滞后的时间变得太长的原因。如果主虚拟机发生故障，则备用虚拟机必须在其上线并开始与外界通信之前，通过回放来“追赶”已经应答的所有日志条目。完成回放的时间基本上是故障点的执行滞后时间，因此备份上线的时间大约等于故障检测时间加上当前执行滞后时间。因此，我们不希望执行延迟时间过长(超过一秒),因为这会增加故障转移时间。

因此，我们还有一个附加的机制可以减慢主 VM 的速度，以防止备份 VM 落后太远。在用于发送和确认日志条目的协议中，我们发送其他信息以确定主虚拟机和备用虚拟机之间的实时执行滞后。通常，执行滞后小于 100 毫秒。如果备份虚拟机开始出现严重的执行延迟（例如，超过 1 秒），则 VMware FT会通过通知调度程序为其分配较少的 CPU（最初只是几个百分点）来降低主虚拟机的速度。我们使用缓慢的反馈循环(we use a slow feedback loop)， 该循环将尝试逐步为主要 VM 确定正确的 CPU 限制，以使备用VM 能够与其执行相匹配。如果备份VM继续落后，我们将继续调低CPU的速度。相反，如果备份 VM 赶上来，我们将逐渐增加主VM 的 CPU 性能，直到备份 VM 恢复到稍有滞后。

请注意，主 VM 的这种速度下降非常罕见，通常仅在系统承受极大压力时才会发生。第 5 节的所有性能数据均包括此类减速的成本。

3.3 Operation on FT VMs

另一个实际问题是处理可能应用于主 VM 的各种控制操作。例如，如果主VM已明确关闭电源，则备用VM也应停止，并且不要尝试上线。再举一个例子，主VM上的任何资源管理更改（例如增加的 CPU 份额）也应应用于备份VM。对于这些类型的操作，将在从主VM到备份VM的日志记录通道上发送特殊的控制条目，以对备份VM执行适当的操作。

通常，应仅在主 VM 上启动 VM 上的大多数操作。然后， VMware FT 发送任意必要的控制条目以对备份 VM 进行适当的更改。VMotion 是唯一可以在主虚拟机和备份 VM 上独立完成的操作。也就是说，主虚拟机和备用虚拟机可以独立地通过VMotion 迁移到其他主机。请注意，VMware FT 确保两个虚拟机都不移动到另一个虚拟机所在的服务器，因为那种情况将不再提供容错能力。

主 VM 的 VMotion 与普通 VMotion 相比增加了一些复杂性， 因为备份 VM 必须在适当的时间与源主 VM(source primary) 断开连接并重新连接到目标主 VM(destination primary VM)。备份 VM 的 VMotion 也有类似的问题，但是增加了额外的复杂性。对于正常的 VMotion，我们要求所有未完成的磁盘 IO 都被静默(即完成),就像在 VMotion 上进行最终切换一样。对于主虚拟机，可以通过等待物理 IO 完成并将这些完成交付给虚拟机来轻松处理这种静默。但是，对于备份 VM，没有简单的方法可以使所有的IO能够在需要的时间点完成，因为备份 VM 必须回放主 VM 的执行并在同一执行点完成 IO。主 VM 可能在正常执行期间始终有磁盘 IO 在运行的工作负载中。VMware FT 具有解决此问题的独特方法。当备份 VM 处于 VMotion 的最终切换点时，它会通过日志记录通道请求主 VM 暂时停止其所有 IO。然后，备份虚拟机的 IO 自然会在单个执行点上被静默，因为它会回放主VM对静默操作的执行。

3.4 Implementation Issue for Disk IOs

存在许多与磁盘 IO 相关的细微实现问题。首先，由于磁盘操作是非阻塞的所以可以并行执行，同时访问同一磁盘位置的磁盘操作可能导致不确定性。另外，我们在磁盘 IO 的实现中直接使用 DMA 往返虚拟机的内存，因此同时访问同一内存页面的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 争用（这种情况很少见），并迫使此类竞速磁盘操作在主磁盘和备份磁盘上以相同的方式顺序执行。

其次，磁盘操作也可能与VM中的应用程序(或 OS) 的内存访问竞争，因为磁盘操作直接通过 DMA 访问 VM 的内存。例如，如果虚拟机中的应用程序/操作系统正在读取内存块，同时该块正在进行磁盘读取，则可能会出现不确定的结果。这种情况也是不太可能的，但是如果发生，我们必须对其进行检测并加以处理。一种解决方案是在作为磁盘操作目标的页面上临时设置页面保护。如果 VM 恰巧访问也是未完成磁盘操作的目标的页面，则页面保护会导致陷阱，并且 VM 可以暂停直到磁盘操作完成。因为更改页面上的 MMU 保护是一项昂贵的操作，所以我们选择使用回弹缓冲区 (bounce buffers)。回弹缓冲区 是一个临时缓冲区，其大小与磁盘操作所访问的内存相同。 磁盘读取操作被修改为将指定的数据读取到缓冲区，并且只有在完成输入输出后，数据才被复制到guest memory。类似地， 对于磁盘写操作，首先将要发送的数据复制到回弹缓冲区， 然后修改磁盘写操作以从回弹缓冲区写入数据。使用反弹缓 冲区可能会减慢磁盘操作，但我们尚未看到它会导致任何明显的性能损失。

第三，当发生故障时，主服务器上存在一些与磁盘IOs相关的未解决(即未完成)问题，备份VM会接管这些问题。新升级的主VM无法确定磁盘操作系统是否已发布到磁盘或成功完成。此外，因为disk IOs不是在备份VM上从外部发出的，所以新提升的主 VM继续运行时，它们不会有明确的IO完成，这最终会导致VM中的guest OS启动一个中止或重置的过程。我们可以发送一个error completion来标记每个 IO 失败，因为即使 IO 成功完成也可以返回一个错误。但是，guest OS可能无法很好地响应其本地磁盘上的错误。相反，我们会在备份虚拟机的上线过程中重新发布挂起的操作系统。因为我们已经消除了所有竞争，并且所有的IOs都直接指定了哪些内存和磁盘块被访问，所以即使这些磁盘操作已经成功完成(即它们是幂等的)，它们也可以被重新发布。

3.5 Implementation Issue for Network IO

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管理程序异步更新虚拟机网络设备的状态。例如，虚拟机执行时，管理程序可以直接更新接收缓冲区。 不幸的是，这些对虚拟机状态的异步更新增加了不确定性。 除非我们可以保证所有更新都在主VM和备份VM的指令流中同时发生，否则备份VM的执行可能与主VM的执行有所不同。

FT 的网络仿真代码的最大变化是禁用了异步网络优化。已修改了使用传入数据包异步更新 VM 环形缓冲区的代码，以强制 guest 虚拟机捕获到虚拟机管理程序，在虚拟机管理程序中，它可以记录更新，然后将更新应用到 VM。类似地，对于FT，通常会以异步方式将数据包异步地从传输队列中拉出的代码被禁用，通常是通过陷阱向虚拟机管理程序进行传输（如下所述）。

消除网络设备的异步更新以及第 2.2 节中描述的延迟发送数据包，给网络性能带来了一些挑战。在运行 FT 时，我们采用了两种方法来提高 VM 网络性能。首先，我们实施了集群优化，以减少虚拟机陷阱和中断。当 VM 以足够的比特率传输数据时，虚拟机管理程序可以对每组数据包执行一个发送陷阱，而在最佳情况下，则为零陷阱，因为它可以将数据包作为接收新数据包的一部分进行发送。同样，虚拟机管理程序可以通过仅post一组数据包的中断来减少传入数据包的 VM 中断数量。

我们对网络的第二次性能优化涉及减少传输数据包的延迟。如前所述，系统管理程序必须延迟所有传输的数据包，直到它从备份VM中接收到足够的(appropriate也有特殊的意思)日志条目的ack为止。减少传输延迟的关键是减少将日志消息发送到备份VM并获得确认所需的时间。我们在这方面的主要优化涉及确保无需任何线程上下文切换就可以完成发送和接收日志条目以及确认。VMware vSphere 虚拟机管理程序允许向 TCP 堆栈注册功能，只要接收到 TCP 数据，这些功能就会从延迟执行上下文(类似于Linux中的tasklet)中调用。

这使我们能够快速处理备份虚拟机上的任何传入的日志消息以及主VM收到的任何ack，而无需任何线程上下文切换。此外，当主VM将要传输的数据包排入队列时，我们会通过调度延迟执行上下文进行刷新，从而强制对相关输出日志条目进行即时的日志刷新(如 2.2 节所述)。

4. Design Alternatives

在我们实现 VMware FT 的过程中，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法。

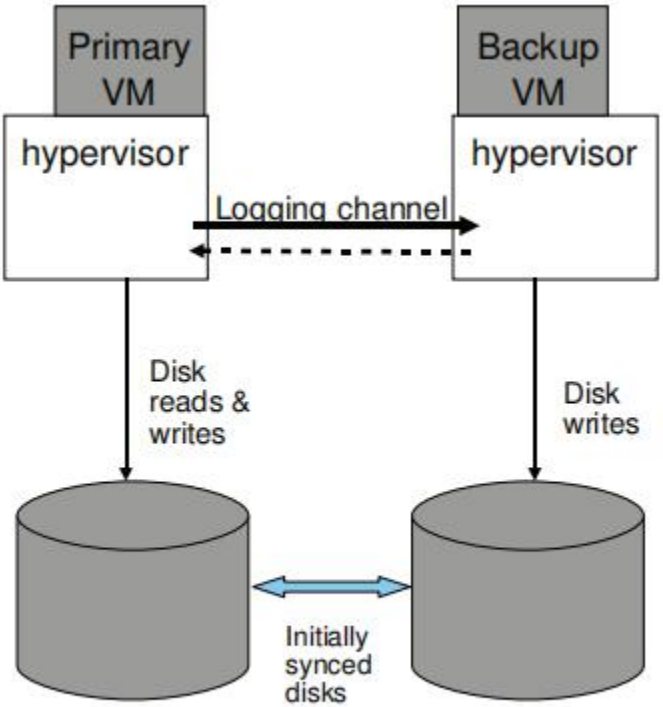


Figure 4: FT Non-shared Disk Configuration.

4.1 Shared vs. Non-shared Disk

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，共享磁盘的内容自然是正确的，并且在发生故障转移时可用。本质上，共享磁盘被认为是主虚拟机和备份虚拟机的外部环境，因此对共享磁盘的任何写入都被视为与外部环境的通信。因此，只有主 VM 才会实际写入磁盘，并且必须根据输出规则延迟写入共享磁盘。

另一种设计是让主虚拟机和备份虚拟机具有单独的(非共享)虚拟磁盘。在此设计中，备份 VM 确实将所有磁盘数据写入其虚拟磁盘，并且这样做自然使其虚拟磁盘的内容与主 VM 的虚拟磁盘的内容保持同步。图 4 说明了此配置。对于非共享磁盘，虚拟磁盘本质上被视为每个 VM 内部状态的一部分。因此， 不必根据输出规则延迟主磁盘的磁盘写入。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。可能是因为共享存储不可用或太昂贵，或者因为运行主VM和备份VM的服务器相距很远（“长距离 FT”）。非共享设计的一个缺点是，非共享设计的一个缺点是，当首次启用容错时，虚拟磁盘的两个副本必须以某种方式明确同步。此外，磁盘在发生故障后可能会不同步，因此发生故障后重新启动备份 VM 时，必须显式重新同步它们。也就是说，FT VMotion 不仅必须同步主VM和备份 VM 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裂脑情况。在这种情况下，系统可以使用其他一些外部决胜者， 例如两个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，则系统可以选择使用基于集群成员多数算法。在这种情况下，仅当 VM 在包含大部分原始节点的通信子集群的一部分的服务器上运行时，才允许 VM 上线。

4.2 Executing Disk Reads on the Backup VM

在我们的默认设计中，备份 VM 永远不会从其虚拟磁盘（共享或非共享）中读取数据。由于磁盘读取被视为输入，自然会通过日志记录通道将磁盘读取的结果发送到备份 VM。

另一种设计是让备份 VM 执行磁盘读取，从而消除对磁盘读取数据的记录。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，这种方法有许多微妙之处。这可能会降低备份VM的执行速度，因为备份VM必须执行所有磁盘读取，并且在到达虚拟机执行点时(在主虚拟机上完成)，如果这些读取没有物理完成，则必须等待。

另外，必须完成一些额外的工作来处理失败的磁盘读取操作。如果主VM读取的磁盘成功，但是备份读取的相应磁盘失败，则必须重试备份读取的磁盘，直到成功为止，因为备份VM必须在内存中获取与主VM相同的数据。相反，如果主VM读取的磁盘发生故障，则必须通过日志记录通道将目标内存的内容发送到备份VM，因为内存的内容将不确定，并且不一定由备份 VM 读取的成功磁盘进行复制。

最后，如果此磁盘读取替代方案与共享磁盘配置一起使用， 则有一个细微之处。如果主 VM 读取了特定磁盘位置，然后很快又写入了相同的磁盘位置，则必须将磁盘写入延迟，直到备份 VM 执行了第一次磁盘读取为止。可以正确检测并处理这种依赖性，但是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会导致实际应用程序的吞

吐量略有下降(1-4%)，但也会显著降低日志记录带宽。因此，在日志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会很有用。

5. Performance Evaluation

在本节中，我们针对VMware FT 的大量应用程序工作负载和网络基准(networking benchmarks)的性能进行了基本评估。为了获得这些结果，我们在相同的服务器上运行主虚拟机和备份虚拟机，每个服务器都具有八个 Intel Xeon 2.8 Ghz CPU 和 8 GB RAM。服务器通过 10 Gbit / s 的交叉网络连接，尽管在所有情况下都可以看到，使用的网络带宽远远小于 1 Gbit / s。两台服务器都从通过标准4gb/s光纤通道网络连接的EMC Clariion访问它们的共享虚拟磁盘。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如下。SPECJbb2005是一个行业标准的Java应用程序基准测试，它占用大量的CPU和内存，几乎不执行任何IO操作。内核编译是运行Linux内核编译的工作负载。由于许多编译过程的创建和销毁，这种工作负载会进行一些磁盘读写，并且非常耗费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。该工作负载执行大量磁盘和网络IO，并且有80个同时进行的数据库会话。微软数据库是一个工作负载，其中微软服务器2005数据库是由数据库基准驱动的，它有16个同步客户端。

	performance (FT / non-FT)	logging bandwidth
SPECJbb2005	0.98	1.5 Mbits/sec
Kernel Compile	0.95	3.0 Mbits/sec
Oracle Swingbench	0.99	12 Mbits/sec
MS-SQL DVD Store	0.94	18 Mbits/sec

Table 1: Basic Performance Results

	base bandwidth	FT bandwidth	logging bandwidth
Receive (1Gb)	940	604	730
Transmit (1Gb)	940	855	42
Receive (10Gb)	940	860	990
Transmit (10Gb)	940	935	60

Table 2: Performance of Network Transmit and Receive to a Client (all in Mbit/s) for 1Gb and 10Gb Logging Channels

5.1 Basic Performance Results

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未在同一 VM 上启用 FT 时性能的比率。计算性能比，如果该值

小于 1 表示 FT 工作负载较慢。显然，在这些代表性工作负载上启用 FT 的开销不到 10％。SPECJbb2005 完全受计算限制，没有空闲时间，但性能良好，因为它具有除计时器中断之外的最少不确定性事件。其他工作负载执行磁盘IO 并具有一定的空闲时间，因此 FT VM 的空闲时间较少可能掩盖了某些 FT 开销。但是，总的结论是 VMware FT 能够以较低的性能开销支持容错 VM。

在表的第三列中，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，记录带宽相当合理，并且 1 Gbit/s网络很容易满足。实际上，低带宽要求表明多个 FT 工作负载可以共享同一 1 Gbit/s网络，而不会产生任何负面性能影响。

对于运行常见guest操作系统(例如 Linux 和 Windows)的VM，我们发现guest OS空闲时的典型日志记录带宽为0.5-1.5 Mbits / sec。“空闲”带宽主要是记录计时器中断的传递的结果。对于具有活动工作负载的 VM，日志记录带宽由必须发送到备份的网络和磁盘输入（接收的网络数据包和从磁盘读取的磁盘块）决定。因此，对于具有非常高的网络接收或磁盘读取带宽的应用程序，日志记录带宽可能比表1中测量的带宽高得多。对于此类应用程序，尤其是在日志记录通道还有其他用途的情况下,日志记录通道的带宽可能会成为瓶颈。

许多实际应用程序在日志记录通道上所需的带宽相对较低，这使得基于回放的容错对于使用非共享磁盘的远程配置非常有吸引力。对于主设备和备用设备可能相隔 1-100 公里的长距离配置，光纤可以轻松支持 100-1000 Mbit / s 的带宽，而等待时间则小于 10 毫秒。对于表 1 中的应用，100-1000 Mbit / s 的带宽应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额外的往返延迟可能会导致网络和磁盘输出延迟20毫秒(最多)。长距离配置仅适用于其客户端可以承受每个请求这样的额外延迟的应用程序。

对于两个磁盘最密集的应用程序，我们已经测量了在备份 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相比对性能的影响。对于 Oracle Swingbench，在备份 VM 上执行磁盘读取时，吞吐量大约降低了 4％。对于 MS-SQL DVD Store，吞吐量大约降低 1％。同时，Oracle Swingbench 的日志记录带宽从 12 Mbits / sec 降低到 3 Mbits / sec，而 MS-SQL DVD Store 从 18 Mbits / sec 降低到 8 Mbits / sec。显然，对于磁盘读取带宽更大的应用程序，带宽节省可能更大。如第 4.2 节所述，预计在备份 VM 上执行磁盘读取时，性能可能会稍差。但是，对于日志记录通道的带宽受限的情况(例如，长距离配置),在备份VM 上执行磁盘读取可能会很有用。

5.2 Network Benchmarks

由于许多原因，网络基准对于我们的系统而言可能是非常具有挑战性的。首先，高速网络可能具有很高的中断率，这要求以很高的速率记录和回放异步事件。第二，以高速率接收数据包的基准测试将导致较高的日志记录流量，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VM。第三，发送数据包的基准测试将受输出规则的约束，该规则会延迟网络数据包的发送，直到收到来自备份的适当确认为止。这种延迟会增加客户端的测量延迟。这种延迟还会降低客户端的网络带宽，因为随着往返延迟的增加，网络协议(如TCP)可能不得不降低网络传输速率。

表 2 给出了根据标准 netperf benchmark进行的许多测试的结果。在所有这些测试中，客户端 VM 和主 VM 通过 1 Gbit / s 网络连接。当主VM和备用VM通过 1 Gbit / s 日志记录通道连接时，前

两行提供发送和接收性能。当主VM和备用VM通过 10 Gbit / s 记录通道连接时，第三和第四行给出了发送和接收性能，该通道不仅具有更高的带宽，而且比 1 Gbit / s 的网络具有更低的延迟。作为粗略的度量，用于 1 Gbit / s 连接的虚拟机管理程序之间的 ping 时间大约为 150 微秒，而用于 10 Gbit / s 连接的 ping 时间大约为 90 微秒。

如果未启用 FT，则主 VM 可以达到接近 1 Gbit / s 的传输和接收线路速率(940 Mbit / s)。为接收工作负载启用 FT 后，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，日志记录通道可能成为瓶颈，如 1 Gbit / s 日志记录网络的结果所示。对于 10 Gbit / s 的日志网络，其影响要小得多。为传输工作负载启用 FT 时，不记录传输数据包的数据，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。总体而言，我们看到 FT 可以在很高的发送和接收速率下显着限制网络带宽，但是仍然可以实现很高的绝对速率。

6. Related Work

Bressoud 和 Schneider [3]描述了为虚拟机实现容错的最初思想(虚拟机是完整包含在管理程序级别的软件)。他们展示了通过带有 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，它们无法实现完全安全的隔离虚拟机。同样，他们没有实现任何故障检测方法，也没有试图解决第 3 节中描述的任何实际问题。更重要的是，他们对 FT 协议施加了许多不必要的约束。首先，他们强加了epochs(时代)的概念，其中异步事件被延迟到设置的时间间隔结束。epochs的概念是不必要的:他们可能已经强加了epochs，因为它们不能有效地回放单个异步事件。其次，他们要求主虚拟机基本上停止执行，直到备份已收到并确认所有先前的日志条目为止。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]描述了一种在操作系统（Unixware）中实现容错的系统，因此该操作系统为该系统上运行的所有应用程序都提供了容错功能。系统调用接口成为必须确定性复制的一组操作。这项工作与基于虚拟机管理程序的工作有相似的限制和设计选择。

Napper 等。[9]和 Friedman 和 Kama [7]描述了容错 Java 虚拟机的实现。在发送有关以下内容的信息时，他们遵循与我们相似的设计，在日志通道上发送关于输入和非确定性操作的信息。像 Bressoud 一样，它们似乎并不专注于检测故障并在故障后重新建立容错能力。此外，它们的实现仅限于为 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，但是要求所有数据都由锁正确保护，或者在访问共享内存时强制执行序列化。

Dunlap 等。[6]描述了针对半虚拟化系统上的调试应用软件的确定性回放的实现。我们的工作支持在虚拟机中运行的任意操作系统，并为这些 VM 实现容错支持，这需要更高级别的稳定性和性能。

Cully 等。[5]描述了一种支持容错 VM 的替代方法,该方法在名为Remus 的项目中被实现。通过使用这种方法，主虚拟机的状态在执行过程中被反复检查，并被发送到收集检查点信息的备份服务

器。因为外部输出有延迟,所以checkpoint必须被频繁地执行(每秒多次)，直到发送并确认下一个checkpoint。这种方法 的优点是，它适用于单处理器和多处理器 VM。主要问题在于，此方法具有很高的网络带宽要求，以便在每个检查点将增量的变化发送到内存状态。[5]中提出的 Remus 的结果显示， 当尝试使用 1 Gbit / s 网络连接每秒传输 40 个检查点以传输内存状态更改时，内核编译和 SPECweb 基准测试的速度降低了 100％至 225％。有许多优化措施可用于减少所需的网络带宽，但尚不清楚通过 1 Gbit / s 的连接能否获得合理的性能。相比之下，我们基于确定性回放的方法可以实现不到 10％ 的开销，并且对于几个实际应用程序，主主机和备用主机之 间的带宽要求小于 20 Mbit / s。

7. CONCLUSION AND FUTURE WORK

我们已经在 VMware vSphere 中设计并实现了一个高效而完整的系统，该系统可为群集中的服务器上运行的虚拟机提供容错（FT）。我们的设计是利用一台备份VM使用VMware确定性回放来复制主VM的执行。如果运行着的主VM发生故障，则备份VM立即接管，而不会造成数据中断或丢失。

总体而言，在商用硬件上的 VMware FT 下，容错 VM 的性能非常出色，对于某些典型应用程序而言，其开销不到 10％。VMware FT 的大部分性能开销都来自使用 VMware 确定性回放以使主VM和备份 VM 保持同步。因此，VMware FT 的低开销源于 VMware 确定性回放的效率。此外，保持主VM和备份VM同步所需的日志记录带宽通常很小，通常小于 20 Mbit / s。由于日志记录带宽在大多数情况下很小， 因此在主VM和备份VM之间相隔很长距离（1-100 公里）的情况下实现两者配置几乎是可行的。因此，在整个站点都出现故障的情况下，VMware FT也可以用于防灾。值得注意的是，日志流通常是可压缩的，并且简单的压缩技术可以以很少的额外 CPU 开销来显着减少日志记录带宽。

我们在 VMware FT 上的结果表明，可以在确定性回放的基础上高效构建容错 VM。这样的系统可以透明地为任何操作系统和应用程序的虚拟机提供容错，并且开销最小。但是，要使容错 VM 系统对客户有用，它还必须健壮，易于使用且高度自动化。一个可用的系统除了复制执行 VM 之外还需要许多其他组件。尤其，通过在本地集群中部署适当的服务器并在该服务器上创建新的备份虚拟机，VMware FT会在出现故障后自动恢复冗余。通过解决所有必要问题，我们展示了一种可用于客户数据中心中实际应用的系统。

通过确定性重放实现容错的一个缺点是，目前确定性回放只对单处理器虚拟机有效。然而目前处理器不断变得越来越强大,单处理器虚拟机对于各种各样的工作负载来说绰绰有余。此外，许多工作负载可以通过使用多个单处理器虚拟机进行扩展，而不是通过使用一个更大的多处理器虚拟机进行扩展。多处理器虚拟机的高性能回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回放。

将来，我们也有兴趣扩展我们的系统来处理部分硬件故障。部分硬件故障是指服务器部分丧失功能或冗余，不会导致数据损坏或丢失。一个例子是与虚拟机的所有网络连接丢失，或者物理服务

器中的冗余电源丢失。如果运行主VM的服务器发生部分硬件故障，在许多情况下(但不是全部)，立即故障切换到备份VM将是有利的。这种故障切换可以立即恢复关键虚拟机的完整服务，并确保虚拟机快速脱离可能不可靠的服务器。