设计一套实用的容错虚拟机

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

摘要

我们实施了一个商用企业级系统，用于提供容错虚拟机，其基础是通过另一台服务器上的备份虚拟机复制主虚拟机（VM）的执行。我们在 VMware vSphere 4.0 中设计了一套完整的系统，该系统易于使用，可在商品服务器上运行，通常可将实际应用程序的性能降低 10%以下。此外，在几个实际应用中，保持主虚拟机和辅助虚拟机同步执行所需的数据带宽不到 20 Mbit/s，这就为在更远的距离上实现容错提供了可能。一个易于使用、可在故障后自动恢复冗余的商用系统，除了需要复制虚拟机执行之外，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。在本章中，我们将介绍我们的基本设计，讨论其他设计选择和一些实现细节，并提供微基准和实际应用的性能结果。

1. 引言

实现容错服务器的一种常见方法是主服务器/备份服务器方法[1]，即在主服务器发生故障时，备份服务器始终可以接替主服务器。备份服务器的状态必须始终保持与主服务器几乎一致，这样当主服务器发生故障时，备份服务器就能立即接管，同时对外部客户隐藏故障，不会丢失任何数据。在备份服务器上复制状态的一种方法是将主服务器的所有状态变化（包括 CPU、内存和 I/O 设备）几乎不间断地传送到备份服务器。但是，发送这些状态（尤其是内存中的变化）所需的带宽可能非常大。

复制服务器的另一种方法有时被称为状态机方法 [13]，这种方法使用的带宽要少得多。其思路是将服务器建模为确定性状态机，通过从相同的初始状态启动服务器并确保服务器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是确定的，因此必须使用额外的协调来确 保 主服务器和备份服务器保持同步。不过，保持主用和备用同步所需的额外信息量远远小于主用状态（主要是内存更新）的变化量。

实施协调，确保确定性退出

物理服务器的执行[14]很困难，尤其是随着处理器频率的提高。相比之下，运行在管理程序之上的虚拟机（VM）是实现状态机方法的绝佳平台。虚拟机可以被视为一个定义明确的状态机，其操作就是被虚拟化的机器（包括其所有设备）的操作。与物理服务器一样，虚拟机也有一些非确定性操作（如读取时间时钟或发送中断），因此必须向备份发送额外信息，以确保其保持同步。由于管理程序可以完全控制虚拟机的执行，包括所有输入的交付，因此管理程序能够捕捉到主虚拟机上有关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，状态机方法可在商品硬件上为虚拟机实施，无需修改硬件，从而可立即为最新的微处理器实施容错。此外，状态机方法所需的带宽较低，因此主备之间的物理间隔可能更大。例如，经过备份的虚拟机可以在分布于校园各处的物理机上运行，这比在同一栋楼内运行的虚拟机更可靠。

我们在 VMware vSphere 4.0 平台上使用优先/备份方法实现了容错虚拟机，该平台可高效运行完全虚拟化的 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。让我们能够记录主程序的执行情况并确保备份程序执行相同程序的基础技术被称为确定性重放技术 [15]。VMware vSphere Fault Tolerance（FT）以确定性重放为基础，但增加了必要的额外协议和功能，以构建完整的容错系统。除了提供硬件容错外，我们的系统还能在故障发生后自动恢复冗余，在本地群集的任何可用服务器上启动新的备份虚拟机。目前，确定性重放和 VMware FT 的生产版本都只支持单处理器虚拟机。记录和重播多处理器虚拟机的执行过程仍在进行中，由于对共享内存的几乎每次访问都可能是非确定性操作，因此存在严重的性能问题。

Bressoud 和 Schneider[3]描述了一个原型实施

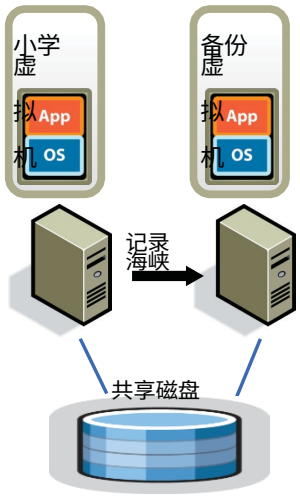

图 1：基本 FT 配置。

在惠普 PA-RISC 平台上开发容错虚拟机。我们的方法与之类似，但出于性能方面的考虑，我们做了一些根本性的改动，并对一些设计方案进行了研究。此外，我们还必须设计和实施系统中的许多附加组件，并处理许多实际问题，以建立一个高效且可供运行企业应用程序的客户使用的完整系统。与所讨论的大多数其他实用系统类似，我们只尝试处理故障停止故障[12]，即在故障服务器导致不正确的外部可见操作之前就能检测到的服务器故障。

本文接下来的内容安排如下。首先，我们描述了我们的基本设计，并详细介绍了我们的基本原理，这些原理可确保在主虚拟机发生故障后由备份虚拟机接管时不会丢失数据。然后，我们将 详细描述建立一个强大、完整和自动化系统所必须解决的许多实际问题。我们还介绍了实现容错虚拟机的几种设计选择，并讨论了这些选择中的权衡问题。接下来，我们给出了我们的实现在一些基准和实际企业应用中的性能结果。最后，我们将介绍相关工作并得出结论。

2. 基本英尺设计

图 1 显示了我们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*备份虚拟机*与主虚拟机保持同步并执行相同的操作，但有少许时间差。我们称这两个虚拟机处于*虚拟同步状态*。虚拟机的虚拟磁盘位于共享存储（如光纤通道或 iSCSI 磁盘阵列）上，因此主虚拟机和备份虚拟机都可以访问这些磁盘进行输入和输出。(我们将在第 4.1 节讨论主虚拟机和备份虚拟机分别拥有非共享虚拟磁盘的设计）。只有主虚拟机会在网络上公布自己的存在，因此所有网络输入都来自主虚拟机。同样，所有其他输入（如键盘和鼠标）都只进入主虚拟机。

主虚拟机接收到的所有输入都会发送到

备份 VM 通过称为*日志通道的*网络连接进行备份。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，还将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。其结果是，备份虚拟机的执行方式始终与主虚拟机相同。不过，管理程序会 放弃备份虚拟机的输出，因此只有主虚拟机才会产生实际输出并返回给客户端。如第 2.2 节所述，主虚拟机和备份虚拟机遵循特定协议，包括备份虚拟机的明确确认，以确保在主虚拟机发生故障时不会丢失数据。

为了检测主用或备用虚拟机是否发生故障，我们的系统结合使用了相关服务器之间的心跳和日志通道上的流量监控。此外，我们必须确保只有一个主 用 或备用虚拟机接管执行，即使出现主用和备用服务器之间失去通信的 "大脑分裂 "情况。

在下面的章节中，我们将详细介绍几个重要方面。在第 2.1 节中，我们将介 绍 确定性重放技术的一些细节，该技术可确保主虚拟机和备份虚拟机通过日志通道发送的信息保持同步。在第 2.2 节中，我们介绍了 FT 协议的基本规则，该规则可确保在主虚拟机发生故障时不会丢失数据。在第 2.3 节中，我们将介绍正确检测和响应故障的方法。

2.1 确定性重放实施

正如我们所提到的，复制服务器（或虚拟机）的执行可以建模为一个确定性状态机的复制。如果两台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。虚拟机的输入范围很广，包括输入的网络数据包、磁盘读取以及键盘和鼠标的输入。非确定性事件（如虚拟启动）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) 正确捕捉所有必要的输入和非确定性，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做到这一点。此外，x86 微处理器中的许多复杂操作都有未定义的副作用，因此也是非确定性的。捕捉这些未定义的副作用并重放它们以产生相同的状态是一项额外的挑战。

VMware 确定性重放 [15] 正是为 VMware vSphere 平台上的 x86 虚拟机提供这种功能。确定性重放将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记录在写入日志文件的日志条目流中。以后可以通过从文件中读取日志条目来精确重放虚拟机的执行。对于非确定性操作，会记录足够的信息，以便以相同的状态变化和输出重现操作。对于非确定性事件，如定时器或 IO 完成时--"....."。

在重放时，也会记录事件发生时的确切指令。在重放过程中，事件将在指令流中的相同点进行交付。VMware 确定性重放实现了一种高效的事件记录和事件交付机制，该机制采用了多种技术，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若干个纪元，其中非确定性事件（如中断）仅在一个纪元结束时发送。由于在中断发生的确切指令上分别发送每个中断的成本太高，因此不划分时间段似乎被用作一种批处理机制。不过，我们的事件交付机制足够高效，因此 VMware 确定性重放无需使用历元。每个中断发生时都会被记录下来，并高效地在
在重放时进行适当的指导。

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情况，但我们并不将日志条目写入磁盘，而是通过日志通道将其发送给备份虚拟机。备份虚拟机会实时重放日志条目，因此其执行与主虚拟机完全相同。不过，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**：如果备份虚拟机在主虚拟机出现故障后接管工作，备份虚拟机将继续以与主 虚拟机向外部世界发送的所有输出完全一致的方式执行。

请注意，在发生*故障转移*（即主虚拟机发生故障后由备份虚拟机接管）后，备份虚拟机开始执行的方式很可能与主虚拟机继续执行的方式大相径庭，因为在执行过程中会发生许多非确定性事件。不过，只要备份虚拟机满足输出要求，在故障切换到备份虚拟机期间就 不会丢失外部可见的状态或数据，客户也不会注意到服务中断或不一致。

可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，直到备份虚拟机收到所有信息，使其至少能重放至输出操作点的执行情况。一个必要条件是，备份虚拟机必须已收到输出操作前生成的所有日志条目。这些日志条目将允许备份虚拟机执行到最后一条日志条目。然而，假设主虚拟机执行输出操作后立即发生故障

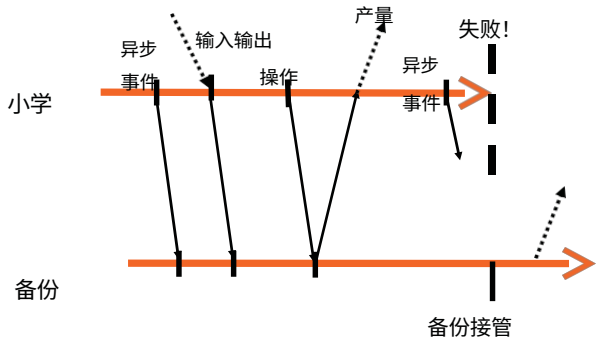

图 2：FT 协议。

。备份虚拟机必须知道，它必须一直重放至输出操作点，并且只能在该点 "上线"（停止重放并接管主虚拟机，如第 2.3 节所述）。如果备份在输出操作前的最后一个日志条目处上线，那么某些非确定性事件（如向虚拟机发送的定时器中断）可能会在其执行输出操作前改变其执行路径。

鉴于上述限制，执行输出要求的最简单方法是在以下位置创建一个特殊日志条目

每个输出操作。然后，输出要求可通过这一特定规则来执行：

> **输出规则**：在备份虚拟机收到并确认与产生输出的操作相关的日志尝试之前，主虚拟机不得向外部世界发送输出。

如果备份虚拟机收到了所有日志条目，包括输出操作的日志条目，那么备份虚拟机就能准确再现主虚拟机在该输出点的状态，因此如果主虚拟机死亡，备份虚拟机将正确达到与该输出一致的状态。反之，如果备份虚拟机在没有收到所有必要日志条目的情况下接手，那么它的状态可能会迅速偏离，从而与主虚拟机的输出不一致。输出规则在某些方面类似于 [11] 中描述的方法，在该方法中，"完全同步 "的 IO 实际上可以被缓冲，只要它在下一次外部通信之前被实际写入磁盘即可。

请注意，"输出规则 "并没有提到要停止主虚拟机的执行。我们只需延迟发送输出，但虚拟机本身可以继续执行。由于操作系统会通过异步中断来指示完成非阻塞网络和磁盘输出，因此虚拟机可以轻松地继续执行，不一定会立即受到输出延迟的影响。相比之下，以前的研究 [3, 9] 通常指出，在进行输出之前，主虚拟机必须完全停止，直到备份虚拟机从主虚拟机确认所有必要信息。

例如，我们在图 2 中展示了一张图，说明了 FT 协议的要求。该图显示了主虚拟机和备份虚拟机上的事件时间线。从主线到备份线的箭头代表日志条目的传输，从备份线到主线的箭头代表确认。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得到确认。如图所示，向外部世界的输出会延迟，直到主虚拟机重新收到备份虚拟机的确认，即它已收到与输出操作相关的日志条目。如果遵循了输出规则，备份虚拟机就能以与主虚拟机上次输出一致的状态接替主虚拟机。

在故障切换情况下，我们无法保证所有输出都能一次性完成。在主服务器打算发送输出时，如果不使用带有两阶段提交的传输操作，备份就无法确定主服务器是在发送最后一次输出之前还是之后崩溃的。幸运的是，网络基础设施（包括常用的 TCP）设计用于处理丢失的数据包和相同（重复）的数据包。需要注意的是，在主设备发生故障时，传入主设备的数据包也可能丢失，因此不会传送到备份设备。但是，传入的数据包可能会因各种与服务器故障无关的原因而丢失，因此网络基础设施、操作系统和应用程序的编写都要确保能够补偿丢失的数据包。

2.3 检测和应对故障

如上所述，如果其他虚拟机出现故障，主虚拟机和备份虚拟机必须迅速做出反应。如果备份虚拟机出现故障，主虚拟机将*进入运行状态*，即离开记录模式（从而停止在日志通道上发送条目）并开始正常执行。如果主虚拟机发生故障，备份虚拟机也应同样*启动*，但过程要复杂一些。由于执行滞后，备份虚拟机可能会有一些已接收并确认的日志条目，但由于备份虚拟机尚未到达执行的适当点，因此尚未被消耗。备份虚拟机必须继续根据日志条目重放其执行过程，直到消耗完最后一个日志条目。此时，备份虚拟机将停止重放模式，开始像正常虚拟机一样执行。实质上，备份虚拟机已晋升为主虚拟机（现在已不再是备份虚拟机）。由于不再是备份虚拟机，当客户操作系统执行输出操作时，新的主虚拟机将向外部世界产生输出。在向正常模式过渡的过程中，可能需要进行一些特定于设备的操作，以使输出正常进行。特别是，为了联网，VMware FT 会自动在网络上公布新的主虚拟机的 MAC 地址，以便物理网络交换机知道新的主虚拟机位于哪台服务器上。此外，新晋升的主虚拟机可能需要重新发出一些磁盘 IO（如第 3.4 节所述）。

有许多可能的方法来检测主虚拟机和备份虚拟机的故障。VMware FT 在运行容错虚拟机的服务器之间使用 UDP 心跳来检测服务器何时可能崩溃。此外，VMware FT 还会监控从主虚拟机发送到备份虚拟机的日志流量，以及从备份虚拟机发送到主虚拟机的确认。由于定时器会定期中断，因此日志流量应该是有规律的，对于正常运行的客户操作系统来说，日志流量永远不会停止。因此，日志条目或确认信息流的停止可能表明虚拟机发生了故障。如果心跳或日志流量停止的时间超过了特定的超时时间（大约几秒钟），就会宣布故障。

然而，任何此类故障检测方法都容易出现 "分脑 "问题。如果备份服务器停止接收主服务器的心跳，这可能表明主服务器已经发生故障，也可能只是意味着仍在运行的服务器之间失去了所有网络连接。如果备份虚拟机上线，而主服务器

如果主虚拟机或备份虚拟机实际上仍在运行，那么数据很可能会损坏，与虚拟机通信的客户端也会出现问题。因此，我们必须确保在检测到故障时，主虚拟机或备份虚拟机中只有一个能正常运行。为避免 "分脑 " 问题，我们利用共享存储来存储虚拟机的虚拟磁盘。当主虚拟机或备份虚拟机要启用时，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，则允许虚拟机上线。如果操作失败，则另一个虚拟机肯定已经上线，因此当前虚拟机实际上会停止运行（"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，那么它就会一直等待，直到可以访问为止。请注意，如果共享存储因存储网络故障而无法访问，那么虚拟机无论如何都可能无法完成有用的工作，因为虚拟磁盘位于同一个共享存储上。因此，使用共享存储解决分脑情况不会带来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，其中一个虚拟机已经上线，VMware FT 会在另一台主机上启动一个新的备份虚拟机，自动恢复冗余。虽然大多数前人的研究都没有涉及这一过程，但它是使容错虚拟机发挥作用的基础，需要精心设计。更多详情请参见第 3.1 节。

3. FT 的实际应用

第 2 节介绍了 FT 的基本设计和协议。然而，要创建一个可用、稳健和自动的系统，还有许多其他组件必须去掉签名并加以实施。

3.1 启动和重启 FT 虚拟机

必须设计的最大附加组件之一是以与主虚拟机相同的状态启动备份虚拟机的机制。在发生故障后重新启动备份虚拟机时，也会用到这种机制。因此，该机制必须适用于处于任意状态（即不只是启动）的正在运行的主虚拟机。此外，我们希望该机制不会严重干扰主虚拟机的执行，因为这将影响虚拟机的任何当前客户。

对于 VMware FT，我们采用了 VMware vSphere 现有的 VMotion 功能。VMware VMotion [10] 允许将运行中的虚拟机从一台服务器迁移到另一台服务器，而且中断时间极短--虚拟机暂停时间通常不到一秒。我们创建了一种经过修改的 VMotion 形式，它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虚拟机。也就是说，我们修改后的 *FT VMotion* 是将虚拟机克隆到远程主机上，而不是将其迁移。FT VMotion 还设置了一个日志记录通道，并使源虚拟机作为主虚拟机进入日志记录模式，而目标虚拟机作为新的备份进入重放模式。与普通 VMotion 一样，FT VMotion 通常会中断主虚拟机的执行，时间不超过一秒。因此，在运行中的虚拟机上启用 FT 是一项简单、无干扰的操作。

启动备份虚拟机的另一个方面是选择运行虚拟机的服务器。容错虚拟机在可以访问共享存储的服务器群集中运行，因此所有虚拟机通常都可以在群集中的任何服务器上运行。这种灵活性降低了 VMware vSphere 恢复 FT 冗余的难度，即使在以下情况下也是如此

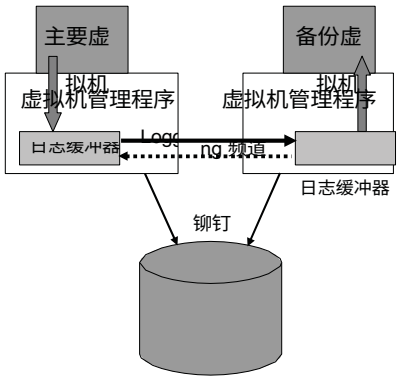

图 3：FT 日志缓冲区和通道。

一个或多个服务器发生故障。VMware vSphere 实施了一个群集服务，用于维护管理和资源信息。当发生故障时，主虚拟机需要一个新的备份虚拟机来重建冗余，这时主虚拟机会通知群集服务它需要一个新的备份。群集服务会根据资源使用情况和其他限制条件确定运行备份虚拟机的最佳服务器，并调用 FT VMotion 来创建新的备份虚拟机。因此，VMware FT 通常可以在服务器发生故障的几分钟内重新建立虚拟机冗余，而容错虚拟机的运 行 不会出现任何明显的中断。

3.2 管理日志记录通道

在管理日志通道上的流量方面，有许多有趣的实施细节。在我们的实施过程中，管理程序会为主虚拟机和备份虚拟机的日志条目维护一个大型缓冲区。当主虚拟机执行时，它会向日志缓冲区生成日志条目，同样，备份虚拟机也会从其日志缓冲区消耗日志条目。主虚拟机日志缓冲区的内容会尽快刷新到日志通道，而日志条目则会在到达后尽快从日志通道读入备份虚拟机的日志缓冲区。备份每次从网络向其日志缓冲区读取一些日志条目时，都会向主设备发送回执。这些回执允许 VMware FT 确定何时可以发送被 "输出规则 "延迟的输出。图 3 展示了这一过程。

如果备份虚拟机在需要读取下一个日志条目时遇到空日志缓冲区，它会停止执行，直到有新的日志条目可用。由于备份虚拟机不对外通信，因此这种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日志缓冲区已满的情况，它必须停止执行，直到日志条目被刷新为止。停止执行是一种自然的流量控制机制，当主虚拟机产生日志条目的速度过快时，它就会放慢速度。但是，这种暂停可能会影响虚拟机的客户端，因为主虚拟机将完全停止并失去响应，直到它可以记录其条目并继续执行。因此，我们的实施必须尽量减少主日志缓冲区填满的可能性。主日志缓冲区可能填满的原因之一是备份虚拟机执行速度太慢，因此消耗日志条目的速度太慢。一般来说，备份虚拟机

必须能够以与主虚拟机记录执行过程大致相同的速度重放执行过程。幸运的是，在 VMware 确定性重放中，记录和重放的开销大致相同。但是，如果托管备份虚拟机的服务器上有大量其他虚拟机（因此资源占用过多），那么尽管备份管理程序的虚拟机调度程序尽了最大努力，备份虚拟机也可能无法获得足够的 CPU 和内存资源，无法以与主虚拟机相同的速度执行。

除了避免在日志缓冲区满时出现意外停顿外，我们还不希望执行滞后过大。如果主虚拟机发生故障，备份虚拟机必须 "迎头赶上"，重放它在上线并开始与外部世界通信之前已经确认的所有日志条目。完成重放的时间基本上就是故障发生时的执行滞后时间，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滞后时间。因此，我们不希望执行滞后时间过长（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虚拟机的速度，以防止备份虚拟机落后太多。在发送和确认日志条目的协议中，我们会发送额外信息，以确定主虚拟机和备份虚拟机之间的实时执行滞后。通常情况下，执行滞后少于 100 毫秒。如果备份虚拟机开始出现明显的执行滞后（例如超过 1 秒），VMware FT 就会通知调度程序稍稍减少主虚拟机的 CPU 使用量（最初仅为百分之几），从而开始降低主虚拟机的运行速度。我们使用一个缓慢的反馈环路，它将尝试逐步确定主虚拟机的适当 CPU 限制，使备份虚拟机的执行速度与其相匹配。如果备份虚拟机继续落后，我们将继续逐步降低主虚拟机的 CPU 限制。反之，如果备份虚拟机迎头赶上，我们就会逐步增加主虚拟机的 CPU 限制，直到备份虚拟机恢复到略微落后的状态。

请注意，主虚拟机的这种减速非常罕见，通常只有在系统承受极大压力时才会发生。第 5 节中的所有性能数据都包含了任何此类减速的成本。

3.3 在 FT 虚拟机上运行

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，如果主虚拟机被明确关闭电源，那么备份虚拟机也应停止运行，而不应试图继续运行。再比如，主虚拟机上的任何资源管理变更（如增加 CPU 共享）也应应用于备份。对于这类操作，会在日志通道上从主设备向备份设备发送特殊的控制条目，以便在备份设备上执行适当的操作。

一般来说，对虚拟机的大多数操作只能在主虚拟机上启动。然后，VMware FT 会发送必要的控制条目，以便在备份虚拟机上进行相应的更改。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 可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、

因为这种情况不再提供容错。

主虚拟机的 VMotion 比普通的 VMotion 增加了一些复杂性，因为备份虚拟机必须断开与源主虚拟机的连接，并在适当的时候重新连接到目标主虚拟机。备份虚拟机的虚拟机移动也有类似的问题，但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 都必须在 VMotion 的最终切换发生时停止（即完成）。对于主虚拟机，只需等待物理 IO 完成并将这些完成的 IO 传递给虚拟机，就 能 轻松实现静态化。但是，对于备份虚拟机来说，由于备份虚拟机必须重放主虚拟机的执行过程，并在同一执行点完成 IO，因此没有简单的方法让所有 IO 在任何要求的点上完成。主虚拟机可能正在运行一种工作负载，在正常执行过程中总是有磁盘 IO 在飞行。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，它会通过日志记录通道请 求 主虚拟机暂时停止其所有 IO。这样，备份虚拟机的 IO 就会在单个执行点上自然而然地 退出，因为它会重播主虚拟机执行退出操作的过程。

3.4 磁盘 IO 的实施问题

磁盘 IO 有许多微妙的实现问题。首先，由于磁盘操作是非阻塞的，因此可以并行执行，同时访问同一磁盘位置的磁盘操作可能导致非确定性。此外，我们的磁盘 IO 实现使用直接往返于虚拟机内存的 DMA，因此同时访问相同内存页的磁盘操作也会导致非确定性。我们的解决方案通常是检测任何此类 IO 竞赛（这种情况很少见），并强制此类竞赛磁盘操作在主备两台虚拟机上以相同的方式按顺序执行。

其次，磁盘操作也可能与虚拟机中应用程序（或操作系统）的内存访问发生竞赛，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/操作系统在读取内存块的同时，磁盘也在读取该内存块，就可能出现非确定性结果。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。如果虚拟机访问的页面同时也是未完成的磁盘操作的目标，页面保护就会导致陷阱，虚拟机就会暂停，直到磁盘操作完成。由于更改 MMU 对页面的保护是一项昂贵的操作，因此我们选择使用*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相同。磁盘读取操作被修改为读取指定数据到反弹缓冲区，只有在 IO 完成后才会将数据复制到客户内存。同样，在磁盘写入操作中，要发送的数据首先被复制到反弹缓冲区，然后磁盘写入被修改为从反弹缓冲区写入数据。使用反弹缓冲区可能会减慢磁盘操作速度，但我们尚未发现它会造成任何明显的性能损失。

第三，当发生故障时，主磁盘上的 IO 未完成（即未完成），由备份接管，这与磁盘 IO 有关。没有办法

新晋升的主虚拟机无法确定磁盘 IO 是否已向磁盘发出或成功完成。此外，由于磁盘 IO 不是在备份虚拟机上从外部发出的，因此在新晋级的主虚拟机继续运行时，不会有明确的 IO 完成，这最终会导致虚拟机中的客户操作系统启动中止或重置程序。我们可以发送错误完成信息，说明每次 IO 都失败了，因为即使 IO 成功完成，返回错误信息也是可以接受的。但是，客户操作系统可能无法很好地响应来自本地磁盘的错误。相反，我们会在备份虚拟机的启动过程中重新发出待处理的 IO。由于我们已经消除了所有竞赛，而且所有 IO 都直接指定了访问的内存和磁盘块，因此即使这些磁盘操作已经成功完成，也可以重新发出（即它们是惰性的）。

3.5 网络 IO 的实施问题

VMware vSphere 为虚拟机网络提供了许多性能优化。其中一些优化基于管理程序异步更新虚拟机网络设备的状态。例如，在虚拟机执行过程中，管理程序可以直接更新再接收缓冲区。遗憾的是，这些对虚拟机状态的异步更新增加了非确定性。除非我们能保证所有更新都发生在主程序和备份程序指令流的同一时刻，否则备份程序的执行可能会偏离主程序的执行。

FT 网络仿真代码的最大变化是禁用了异步网络优化。异步更新虚拟机环形缓冲区与传入数据包的代码已被修改，以强制客户机捕获到管理程序，在管理程序中记录更新，然后将其应用到虚拟机。同样，异步从传输队列中提取数据包的代码在 FT 中也被禁用，而是通过向管理程序发送陷阱来完成传输（下文指出的情况除外）。

取消网络设备的异步更新，再加上第 2.2 节所述的延迟发送数据包，给网络性能带来了一些挑战。我们采取了两种方法来提高运行 FT 时的虚拟机网络性能。首先，我们实施了集群优化，以减少虚拟机陷阱和中断。当虚拟机以足够高的比特率传输数据流时，管理程序可以为每组数据包执行一次发送陷阱，在最佳情况下，陷阱次数为零，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，管理程序只需为一组数据包发布中断，就能减少虚拟机接收数据包的中断次数。

我们对网络性能的第二个优化是减少传输数据包的延迟。如前所述，管理程序必须延迟所有传输的数据包，直到收到备份对适当日志条目的确认。减少传输延迟的关键在于缩短向备份发送日志信息并获得确认所需的时间。我们在这方面的主要优化措施是确保发送和接收日志条目和确认都能在没有任何线程上下文切换的情况下完成。VMware vSphere 虚拟机管理程序允许在 TCP 栈中注册函数，只要接收到 TCP 数据，就会从延迟执行上下文（类似于 Linux 中的 tasklet）中调用这些函数。这种延迟

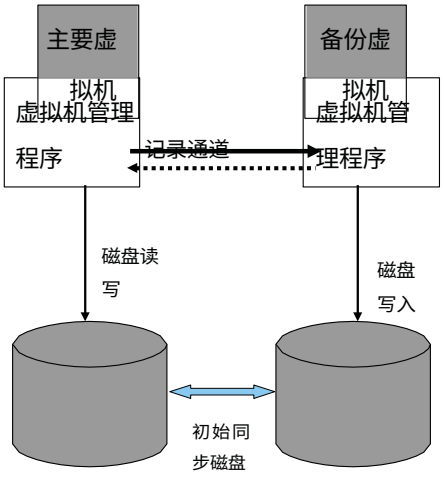

图 4：FT 非共享磁盘配置。

这样，我们就能在不切换任何线程上下文的情况下，快速处理备份上的任何传入日志信息和主虚拟机收到的任何确认信息。此外，当主虚拟机排队等待传输数据包时，我们会通过调度延迟执行上下文来强制立即刷新相关输出日志条目（如第 2.2 节所述）。

4. 设计方案

在实现 VMware FT 的过程中，我们探索了许多有趣的设计方案。在本节中，我们将探讨其中一些替代方案。

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，如果发生故障切换，共享磁盘的内容自然是正确和可用的。从本质上讲，共享磁盘被视为主虚拟机和备份虚拟机的外部磁盘，因此对共享磁盘的任何写入都被视为与外部世界的通信。因此，只有主虚拟机才会对磁盘进行实际写入，而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独立（非共享）的虚拟磁盘。在这种设计中，备份虚拟机将完成对其虚拟磁盘的所有磁盘写入，这样做自然会使其虚拟磁盘的内容与主虚拟机虚拟磁盘的内容保持同步。图 4 展示了这种配置。在非共享磁盘的情况下，虚拟磁盘基本上被视为每个虚拟机内部状态的一部分。因此，主虚拟机的磁盘写入无需根据 "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。出现这种情况的原因可能是共享存储不可用或过于昂贵，或者运行主虚拟机和备份虚拟机的服务器相距甚远（"远距离 FT"）。非共享设计的一个缺点是，在首次启用容错时，必须以某种方式明确同步虚拟磁盘的两个副本。此外，故障发生后磁盘可能会不同步，因此

它们必须在故障后重新启动备份虚拟机时显式地重新同步。也就是说，FT VMotion 不仅要同步主虚拟机和备份虚拟机的运行状态，还要同步它们的磁盘状态。

在非共享磁盘配置中，可能没有可用于处理分脑情况的共享存储。在这种情况下，系统可以使用其他外部平衡器，如两个服务器都能对话的第三方服务器。如果服务器是拥有两个以上节点的集群的一部分，系统也可以使用基于集群成员资格的多数决算法。在这种情况下，只有当虚拟机运行在一个服务器上，而该服务器又是一个包含大多数原始节点的通信子群集的一部分时，虚拟机才会被允许上线。

4.2 在备份虚拟机上执行磁盘读取

在我们的默认设计中，备份虚拟机从不读取其虚拟磁盘（无论是共享磁盘还是非共享磁盘）。由于磁盘读取被视为一种输入，因此自然会通过日志记录通道向备份虚拟机发送磁盘读取的结果。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。对于磁盘读取次数较多的工作负载，这种方法可以大大减少日志记录通道的流量。不过，这种方法也有一些微妙之处。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，并在磁盘读取到达虚拟机中的某个点时等待物理上未完成的磁盘读取。在他们完成初选的地方执行。

此外，还必须做一些额外的工作来处理磁盘读取操作失败的问题。如果主磁盘读取成功，但备份读取相应磁盘失败，则必须重试备份读取的磁盘，直到成功为止，因为备份必须在内存中获得与主磁盘相同的数据。相反，如果主磁盘读取失败，则必须通过日志通道向备份发送目标内存的内容，因为内存内容将无法确定，而且备份虚拟机成功读取磁盘后也不一定会复制内存内容。

最后，在共享磁盘配置中使用磁盘读取替代方案还有一个微妙之处。如果主虚拟机对特定磁盘位置进行读取，随后很快又对同一磁盘位置进行写入，那么磁盘写入必须延迟到备份虚拟机执行完第一次磁盘读取后进行。这种依赖关系可以被检测到并正确处理，但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的吞吐量略有降低（1%-4%），但也会明显减少日志带宽。因此，在日志记录通道带宽相当大的情况下，在备份虚拟机上执行磁盘读取可能会很有用。有限。

5. 绩效评估

在本节中，我们将对 VMware FT 的性能进行基本评估，包括一些应用工作负载和网络基准。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，每台服务器配备 8 个英特尔至强 2.8 Ghz CPU 和 8 Gbytes 内存。服务器通过 10 Gbit/s 交叉网络连接，但正如我们在所有情况下看到的那样，使用的网络带宽远远小于 1 Gbit/s。两台服务器均访问其共享的虚拟

	性能 (全日制/非全日制)	伐木 频带
SPECJbb2005	0.98	1.5 兆位/秒
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

EMC Clariion 的磁盘通过标准 4 Gbit/s 光纤通道网络连接。用于驱动部分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，它的 CPU 和内存密集度非常高，而 IO 却很少。内核编译是一种运行 Linux 内核编译的工作负载。由于需要创建和销毁许多编译进程，因此该工作负载会进行一些磁盘读写操作，CPU 和 MMU 密度非常高。Oracle Swingbench 是一种工作负载，其中 Oracle 11g 数据库由 Swingbench OLTP（在线事务处理）工作负载驱动。该工作负载会产生大量磁盘和网络 IO，并有 80 个并发数据库会话。MS-SQL DVD Store 是由 DVD Store 基准驱动 Microsoft SQL Server 2005 数据库的工作负载，该基准有 16 个并发客户端。

5.1 基本性能结果

表 1 列出了基本性能结果。对于列出的每个应用，第二列给出了在运行服务器工作负载的虚拟机上启用 FT 时的应用性能与在同一虚拟机上未启用 FT 时的性能之比。性能比的计算方法是，数值小于 1 表示 FT 工作负载的速度较慢。显然，在这些代表性工作负载上启用 FT 的开销小于 10%。SPECJbb2005 完全受计算约束，没有空闲时间，但性能很好，因为除了定时器中断外，它的非确定性事件极少。其他工作负载进行磁盘 IO 并有一些空闲时间，因此 FT 虚拟机空闲时间较少这一事实可能会掩盖部分 FT 开销。不过，总的结论是，VMware FT 能够以相当低的性能开销支持容错虚拟机。

在表格的第三列，我们给出了运行这些应用时在日志通道上发送的数据的平均带宽。对于这些应用而言，日志记录带宽相当合理，1 Gbit/s 网络即可轻松满足要求。事实上，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，而不会对性能产生任何负面影响。

	基础 频带	FT 频带	伐木 频带
接收（1Gb）	940	604	730
传输（1Gb）	940	855	42
接收（10Gb）	940	860	990
传输（10Gb）	940	935	60

对于运行 Linux 和 Windows 等常见客户操作系统的虚拟机，我们发现客户操作系统空闲时的典型日志带宽为 0.5-1.5 Mbits/秒。空闲 "带宽主要是记录定时器中断的结果。对于工作负载活跃的虚拟机，日志带宽主要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）所支配。因此，日志带宽可能会大大增加。

表 2：1Gb 和 10Gb 记录通道的网络传输和重新传输到客户端的性能（单位均为 Mbit/s） 表 3：1Gb 和 10Gb 记录通道的网络传输和重新传输到客户端的性能（单位均为 Mbit/s

对于网络接收带宽或磁盘读取带宽非常高的应用，记录通道的带宽可能会高于表 1 中的测量值。对于这类应用，日志通道的带宽可能会成为瓶颈，尤其是在日志通道还有其他用途的情况下。

许多实际应用在日志通道上所需的带宽相对较低，这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。对于主用和备份可能相距 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，延迟时间小于 10 毫秒。对于表 1 中的应用，100-1000 Mbit/s 的带宽足以提供良好的性能。但要注意的是，主备之间的额外往返延迟可能会导致网络和磁盘输出延迟达 20 毫秒。长距离配置只适用于客户可以容忍每次请求都有这样的额外延迟的应用。

对于两个磁盘密集度最高的应用程序，我们已经确定了在备份虚拟机上执行磁盘读取（如第 4.2 节所述）与通过日志通道发送磁盘读取数据对性能的影响。对于 Oracle Swingbench，在备份虚拟机上执行磁盘读取时，吞吐量降低了约 4%；对于 MS-SQL DVD Store，吞吐量降低了约 1%。同时，Oracle Swing- bench 的日志带宽从 12 Mbits/sec 降至 3 Mbits/sec，MS-SQL DVD Store 的日志带宽从 18 Mbits/sec 降至 8 Mbits/sec。显然，对于磁盘读取带宽更大的应用，节省的带宽会更多。如第 4.2 节所述，在备份虚拟机上执行磁盘读取时，预计性能可能会稍差一些。不过，在日志记录通道带宽有限的情况下（如长距离配置），在备份虚拟机上执行磁盘读取可能会很有用。

5.2 网络基准

对我们的系统来说，网络基准测试具有相当大的挑战性，原因有很多。首先，高速网络可能具有非常高的中断率，这就要求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会导致高日志流量，因为所有这些数据包都必须通过日志通道发送到备份。第三，发送数据包的基准会受到 "输出规则 "的限制，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。这种延迟会增加客户端的测量延迟。这种延迟还可能降低客户端的网络带宽，因为网络协议（如 TCP）可能会对客户端的网络带宽造成影响。

随着往返延迟的增加，网络传输速率也会降低。

表 2 列出了我们使用标准 `netperf` 基准进行的一系列测量结果。在所有这些测量中，客户端虚拟机和主虚拟机通过 1 Gbit/s 网络连接。前两行给出了主主机和备份主机通过 1 Gbit/s 日志通道连接时的发送和重接收性能。第三行和第四行是主服务器和备份服务器通过 10 Gbit/s 日志通道连接时的发送和接收性能，10 Gbit/s 日志通道不仅带宽比 1 Gbit/s 网络高，延迟也比 1 Gbit/s 网络低。粗略计算，1 Gbit/s 连接的管理程序之间的 ping 时间约为 150 微秒，而 10 Gbit/s 连接的 ping 时间约为 90 微秒。

不启用 FT 时，主虚拟机的发送和接收速率可接近（940 Mbit/s）1 Gbit/s 线路速率。当接收工作负载启用 FT 时，日志带宽会非常大，因为所有进入的网络数据包都必须通过日志通道发送。因此，如 1 Gbit/s 记录网络的结果所示，记录通道可能成为瓶颈。10 Gbit/s 日志网络的影响要小得多。当传输工作负载启用 FT 时，传输数据包的数据不会被记录，但网络中断仍必须被记录。记录带宽要低得多，因此可实现的网络传输带宽要高于网络重接收带宽。总之，我们可以看到，在发送和接收速率非常高的情况下，FT 会极大地限制网络带宽，但仍可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管理程序级别的软件来实现虚拟机容错的最初想法。他们通过使用惠普 PA-RISC 处理器的服务器原型演示了保持备份虚拟机与主虚拟机同步的可行性。但是，由于 PA-RISC 架构的限制，他们无法实现完全安全、隔离的虚拟机。此外，他们没有实施任何故障检测方法，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 FT 协议施加了许多不必要的限制。首先，他们提出了一个 "纪元"（epochs）的概念，即异步事件被延迟到设定的时间间隔结束。纪元的概念是不必要的--他们可能是因为无法高效地重放单个异步事件而强加了这个概念。其次，他们要求主虚拟机停止执行，直到备份收到并确认之前的所有日志条目。然而，只有输出本身（如网络数据包）必须延迟，主虚拟机本身可以继续执行。

Bressoud [4] 描述了一种在操作系统（Unixware）中实现容错的系统，并因此为在该操作系统上运行的所有应用程序提供容错。系统调用接口成为必须确定性复制的操作集。这项工作与基于管理程序的工作具有类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 虚拟机的实施情况。他们采用了与我们类似的设计，即发送有关

输入和日志通道上的非确定性操作。与 Bressoud 一样，他们似乎并不关注故障检测和故障后重建容错。此外，它们的实施仅限于为在 Java 虚拟机中运行的应用程序提供容错。这些系统试图解决多线程 Java 应用程序的问题，但要么要求所有数据都受到锁的正确保护，要么要求在访问共享内存时执行序列化。

Dunlap 等人[6]描述了一种确定性重放的实现方法，其目标是在准虚拟化系统上调试应用软件。我们的工作支持在虚拟机内运行的任意操作系统，并为这些虚拟机提供容错支持，这要求更高水平的稳定性和性能。

Cully 等人[5]描述了另一种支持容错虚拟机的方法及其在名为 Remus 的项目中的实施。采用这种方法，在执行过程中会反复对主虚拟机的状态进行检查点，并将其发送到备份服务器，由备份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下一个检查点被发送和确认之后。这种方法的优势在于它同样适用于单处理器和多处理器虚拟机。主要问题是，这种方法需要很高的网络带宽，才能在每个检查点发送内存状态的增量变化。文献[5]中介绍的 Remus 结果显示，当尝试使用 1 Gbit/s 网络连接传输内存状态变化时，内核编译和 SPECweb 基准测试的速度降低了 100%到 225%，每秒进行 40 次检查点。有许多优化措施可以降低所需的网络带宽，但目前还不清楚 1 Gbit/s 的连接是否能实现合理的性能。相比之下，我们基于确定性重放的方法可以实现不到 10% 的开销，在几个实际应用中，主主机和备份主机之间所需的带宽不到 20 Mbit/s。

7. 结论与未来工作

我们在 VMware vSphere 中设计并实施了一个高效而完整的系统，可为集群中服务器上运行的虚拟机提供故障容错（FT）。我们的设计基于使用 VMware 确定性重放技术通过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，备份虚拟机会立即接管，不会中断或丢失数据。

总体而言，在商品硬件上使用 VMware FT 的容错虚拟机性能非常出色，对于某些典型应用而言，开销不到 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备虚拟机同步的开销。因此，VMware FT 的低开销来自于 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很小，通常小于 20 Mbit/s。由于大多数情况下日志记录带宽很小，因此似乎可以实现主虚拟机和备份虚拟机相隔很远（1-100 千米）的配置。因此，VMware FT 可用于以下情况

此外，还能防止整个站点发生故障的灾难。值得注意的是，日志流通常是非常容易压缩的，简单的压缩技术只需少量额外的 CPU 开销，就能显著降低日志带宽。

我们在 VMware FT 上取得的成果表明，容错虚拟机的高效实施可以建立在去终结重放的基础上。这种系统能以最小的开销为运行任何操作系统和应用程序的虚拟机提供透明的容错。不过，容错虚拟机系统要想对客户有用，还必须强大、易用和高度自动化。一个可用的系统除了需要复制执行虚拟机外，还需要许多其他组件。特别是，VMware FT 可在故障发生后自动恢复冗余，方法是在本地群集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。通过解决所有必要问题，我们展示了一个可用于客户数据中心实际应用的系统。

通过去终结性重放实现容错的一个折衷方案是，目前确定性重放只针对单处理器虚拟机有效实施。然而，单处理器虚拟机对于各种工作负载来说都绰绰有余，尤其是物理处理器的功能在不断增强。此外、
许多工作负载可以通过使用多个单处理器虚拟机来扩展，而不是通过使用一个较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， 只要微处理器提供一些额外的硬件支持，就有可能实现高性能重放。其中一个研究方向可能是扩展事务内存模型，以促进多处理器重放。

未来，我们还有兴趣扩展我们的系统，以处理部分硬件故障。所谓部分硬件故障，是指服务器部分功能或冗余丢失，但不会导致数据损坏或丢失。例如，虚拟机失去所有网络连接，或物理服务器失去冗余电源。如果运行主虚拟机的服务器发生部分硬件故障，在许多情况下（但并非所有情况），立即向备份虚拟机故障转移是有利的。这样的故障切换可以立即恢复关键虚拟机的全部服务，并确保虚拟机迅速脱离可能不可靠的服务器。

致谢

我们要感谢 Krishna Raja，他提供了许多性能结果。有许多人参与了 VMware FT 的实施。确定性重放（包括支 持 各种虚拟设备）和基础 FT 功能的核心实施者包括 Lan Huang 、 Eric Lowe 、 Slava Malyugin 、 Alex Mirgorodskiy 、 Kaustubh Patil 、 Boris Weissman 、 Petr Van- drovec 和 Min Xu。此外，还有许多其他人员参与了 VMware vCenter 中 FT 的高层管理。Karyn Ritter出色地完成了大部分管理工作。

8. 参考文献

[1] ALSBERG, P. and Day, J. A Principle for Resilient 分布式资源共享》。*第二届软件工程国际会议论文集》*（1976 年），第 627-644 页。

[2] AMD 公司。*AMD64 架构程序员手册》*。加利福尼亚州桑尼维尔。

[3] Bressoud, T., and Schneider, F. Hypervisor-based Fault Tolerance.*SOSP 15*（1995 年 12 月）*论文集*。

[4] Bressoud, T. C. TFT：应用透明容错软件系统》。*第二十八届容错计算国际研讨会论文集》*（1998 年 6 月），第 128-137 页。

[5] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., HUTCHISON, N., and Warfield, A. Remus：通过异步虚拟机复制实现高可用性。*第五届 USENIX 网络系统设计与实现研讨会论文集》*（2008 年 4 月），第 161-174 页。

[6] Dunlap, G. W., KING, S. T., Cinar, S., Basrai, M., and Chen, P. M. ReVirt：通过虚拟机日志和重放实现入侵分析。*2002 年操作系统设计与实现研讨会论文集》*（2002 年 12 月）。

[7] 弗里德曼，R.和卡马，A.透明容错 Java 虚拟机。*可靠分布式系统论文集》*（2003 年 10 月），第 319-328 页。

[8] 英特尔公司。IntelA^□R *64 和 IA-32 体系结构软件开发人员手册》*。加利福尼亚州圣克拉拉市。

[9] Napper, J., Alvisi, L., and Vin, H. A 容错 Java 虚拟机。*可依赖系统与网络国际会议论文集》*（2002 年 6 月），第 425-434 页。

[10] Nelson, M., LIM, B.-H., and HUTCHINS, G. Fast Transparent Migration for Virtual Machines.*2005 年 USENIX 技术年会论文集》*（*2005 年* 4 月）。

[11] NIGHTINGALE, E. B., VEERARAGHAVAN, K., Chen、P.M., and Flinn, J. Rethink the Sync.*2006 年操作系统设计与实现研讨会论文集》*（2006 年 11 月）。

[12] SCHLICTING, R., and Schneider, F. B. Fail-stop Processors：设计容错计算系统的方法》。*ACM Computing Surveys 1*, 3 (Aug. 1983), 222-238.

[13] Schneider, F. B. 使用状态机方法实现容错服务：A tutorial.*ACM Computing Surveys 22*, 4 (Dec. 1990), 299-319.

[14] 斯特拉图斯技术。受益于 Stratus 持续处理技术：适用于微软 Windows 服务器环境的自动 99.999% 正常运行时间。见 http://www.stratus.com/~/media/-Stratus/Files/Resources/WhitePapers/continuous-processing-for-windows.pdf，6 月。2009.

[15] Xu, M., MALYUGIN, V., Sheldon, J., VENKITACHALAM, G., and WEISSMAN, B. ReTrace：利用虚拟机确定性重放收集执行轨迹。*2007 年建模、基准测试和仿真研讨会论文集》*（2007 年 6 月）。