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

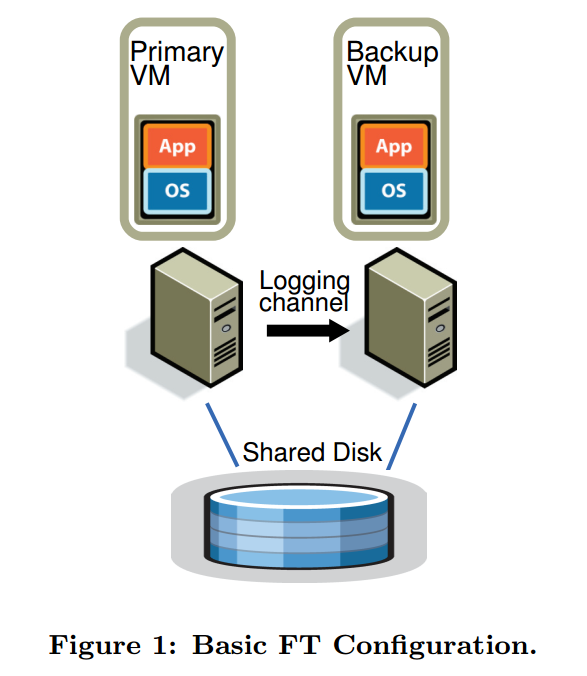


图1展示了我们的容错虚拟机系统的基本配置。对于一个给定的虚拟机，当我们想提供容错（主虚拟机），我们在另外一台物理服务器上运行一个备份虚拟机，它用于保持同步，并且执行和主虚拟机一样的操作，虽然有一个很小时间的延迟。我们称这两个虚拟机处于虚拟一致。这些虚拟机的虚拟磁盘都在共享存储设备上，因此输入输出都可以被主备虚拟机访问。（在4.1章节我们将会讨论一种采用分离不共享虚拟磁盘的主备虚拟机方案）只有主虚拟机在网络中广告它的存在，所以所有的网络输入都会到达主虚拟机。相似地，所有其他的输入（例如键盘和鼠标）也只到达主虚拟机。

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。

## 故障检测与响应

# 容错（FT）的实际实现

## 容错虚拟机（FT VM）的启动和重启

## 日志信道的管理

## 容错虚拟机（FT VM）的操作

## 磁盘IO的实现问题

## 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