Lecture 4 VMware FT

1. 背景知识

1.1. 虚拟机分类（按架构）

- 裸金属架构
- 寄居架构

1.2. 虚拟机分类（按平台）

- 完全虚拟化：不需要更改guest os内核，guest os不知道自己是虚拟机。guest os会产生特权指令，需要VMM（Virtual Machine Monitor）进行指令的翻译，性能差。VMWare、Hyper-V
- 半虚拟化：需要更改guest os的内核，guest os知道自己是虚拟机。特权指令通过超级调用实现。半虚拟化不能虚拟化windows，因为内核不开源无法更改。Xen
- 硬件支持虚拟化：对CPU进行修改，支持两种模式。在虚拟机模式下，guest os可以运行绝大多数的硬件指令
- 软件模拟虚拟化：使用软件模拟出所有的硬件，包括CPU（上面的方法中guest OS均可以直接或间接使用物理CPU）。QEMU

1.3. Docker

- 隔离性比虚拟机更差
- docker共享主机os内核

本论文由VMware的工程师撰写

2. 容错服务器（fault-tolerant server）典型设计

2.1. primary/backup结构：

- 实现方法一：**State Transfer**
  - backup的所有状态（包括CPU，主存，IO设备）都是高度一致的
  - 需要发送完整的内存数据

    - 对网络带宽要求很高
    - 但是似乎更适用于多核/多处理器的机器
- 实现方法二：**状态机（state machine）**
    - 确保primary和backup一开始都在相同的初始状态，两台机器以相同的顺序接受相同的确定性输入请求
    - 由于有些请求不是确定性的，因此需要额外的步骤来进行同步，但是代价远远小于方法一
    - 这种方法随着CPU频率的提高，实现起来越来越困难。只适用于单处理器。多处理器之间指令的交织会导致不确定性
    - 当primary fail切换至backup时，无法做到让外界察觉不到这个转换，因此必须要对client造成异常（anomaly）来通知client，原始primary VM已经故障，需要与另一台server通信

2.2. 虚拟机

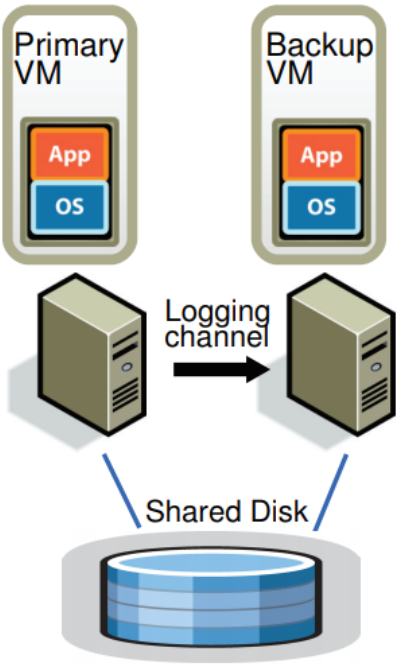
- 可以实现primary/backup结构的状态机方式，换言之，虚拟机⊆状态机
- 运行在超级监管程序之上（hypervisor）
- 虚拟机是状态机⇒ 虚拟机（状态机）的操作 = 虚拟机之上被虚拟化的机器的操作
- 虚拟机也会有**非确定化的操作（例如中断，读取时钟周期计数器等操作）**。但hypervisor具有对虚拟机的完全控制，因此也能获得非确定化操作的所有信息，可以使用这个信息在primary虚拟机和backup虚拟机之间进行备份
- 虚拟机之间备份的带宽要求非常小，这一特性决定了使用虚拟机进行primary/back备份可以使得primary和back具有更好的物理隔离性。因此可以使用分布式虚拟化管理：即不同的虚拟机可以运行在不同的机器上
- **Deterministic Replay （确定性重放）**：一种允许记录primary执行，并且使得backup进行相同执行操作的技术
    - 本论文的设计基于确定性重放，在此基础上增加了额外的协议和功能
    - **确定性执行**：
        - 时间确定性：规定时间内完成
        - 数据确定性：相同数据集产生相同的结果

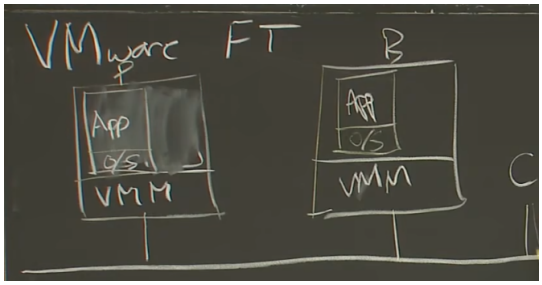
2.3. 原理

2.3.1. 相关概念

- **fail-stop failure**：服务器故障。这些故障可以在服务器造成错误的行为之前被检测出来。不包括应用程序的代码错误，硬件本身的设计错误等等
- **virtual lockstep**：primary和bakcup的一种状态。primary和backup的执行是同步的，完全相同的，并且backup的执行有略微延迟的**一种状态**
- **failover**：primary故障而后backup接管的这一事件
- **go live**：正常执行。一方面可能是由于primary发现backup故障，因此不等待backup回复acknowledgement，退出record mode，直接正常执行；另一方面也可能是由于failover，backup接管。

2.3.2. 系统结构





1. 本论文进行复制的级别（粒度）：本论文涉及的复制几乎是最底层的**机器级别**，完全保证primary和backup是同步的，甚至都要保证中断发生的位置相同。而GFS此类系统，则在**应用**

程序级别进行复制，只会通过chunk和chunk handler来进行复制，而不管机器如何运行

2. primary和backup运行在物理上隔离的虚拟机上
3. primary与backup共用磁盘，通过光纤信道等技术实现。共用存储器上有每个VM的虚拟磁盘
4. 只有primary才会应答client，网络包发送过程需要经过VMM，VMM会识别backup，然后将backup的输出丢弃。而对于磁盘输出，则primary和backup都会写入磁盘。
5. 只有primary才会收到input（包括网络包和磁盘读取的数据），并且获得外设。primary会通过一种叫做**logging channel**（专指primary和backup之间为了传递log entry的网络通路）将input发给backup。结构如图

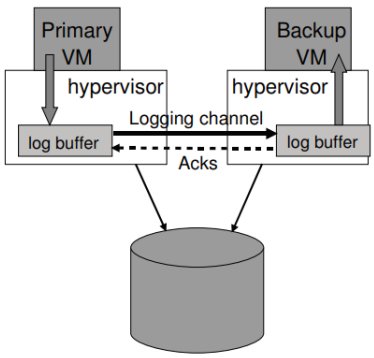


Figure 3: FT Logging Buffers and Channel.

- hypervisor为每个VM都实现了一个**log buffer**
- primary向log buffer写入log entry，backup从primary的log buffer中读取，通过logging channel传递到自己的log buffer中
- 对于**每一条**log entry，backup都会向primary发送acks，告知自己已经收到了log entry，等待下一步
- log entry可能的结构：
  - 指令的编号（从启动开始的编号，不是地址）
  - event的类型
    - 中断（network package到达时会引发终端）
    - weird instruction：即那些在不同的时间不同的机器上执行结果可能不同的指令，例如随机数生成指令。
  - 数据。包括网络包的数据、weird指令在priamry之上执行的结果、primary读取磁盘的结果（不允许backup读取磁盘，原因见后文）

- 中断传输细节：primary的物理主机发出定时器中断，VMM在适当的时机停止priamry的运行，得到指令编号，然后再向primary VM发起模拟的定时器中断。而后通过logging channel发送给backup。（运行backup的物理主机也会有定时器中断，但是并不会传递模拟中断到backup VM中。）backup收到条目后，会设置CPU让CPU在到达primary发过来的指令编号时，产生中断。这时，backup VMM会从backup VM处重新获取控制权，而后向backup VM发送模拟的定时器中断。由此，使得backup和primary的中断发生在统一指令处。**注意，这在当时需要改造CPU，但目前为止的CPU已经完全支持这一功能。**

6. log buffer同步问题
   - 当backup向primary索取log entry但primary的log buffer为空时，backup停止执行，等待primary写入log entry
   - 当primary向自己的log buffer写入log entry但log buffer已满时，primary停止执行，直到backup取走log entry（会导致primary无法应答client的请求）
     - primary的log 缓冲区溢出的主要原因是primary生产速度和bakcups消费速度不匹配（backup太慢）
     - 但研究人员发现record（由primary进行）和replay（由backup进行）的速率差不多
     - 但如果运行backup VM的主机工作负载过大，则会导致backup得不到足够的资源（CPU时间，内存使用等）
7. hypervisor会将backup的输出丢弃，只利用primary的输出。但当primary故障时，也有相关的协议可以保证使用backup，并且数据不丢失

2.3.3. 确定性重演

1. 非确定性事件/操作 （non-deterministic events/operations）带来的挑战
   - 正确地捕获所有的输入和非确定性操作/事件
   - 正确地将所有的输入和非确定性操作/事件应用到backup VMs上
   - 确保不会对性能带来太大的负面影响
2. VMware确定性重演会记录一个VM的所有输入和所有可能的非确定性事件，以日志条目流的形式写入日志文件。对于非确定性事件/执行，会记录足够多的数据，例如非确定性事件发生的具体指令
3. 日志不会写入磁盘，会通过logging channel来进行传输
4. 对于输出的操作，会在输出时也产生一条log entry？？？？？？？？？？？？

2.3.4. **启动与重启VM**

需要有一种机制，能够启动backup VM，并且使得backup VM的状态与primary一致。同时，这种机制也可以被用在重启故障的backup VM。

- 设计该机制的注意点：对primary执行的影响要小
- 本论文在VMware VSphere已有的**VMotion**功能进行了改造。原有的VMotion可以支持将一台正在运行的虚拟机从一个服务器**迁移**到另一个服务器，并且最小化对正在运行的VM的影响（正在运行的VM需要暂停，但是暂停时间通常小于1s）。而本论文将VMotion改造成为新的**FT VMtion**功能，可以允许其完全复制一台VM（包括内存，寄存器等）。
- VMotion不会把一台虚拟机复制到一台已经运行了虚拟机的机器上（这种情况下不再具有容错的功能）
- FT VMotion还会建立logging channel，使得source VM进入logging模式（是否等同于record模式？？？？？？？？？？？？？？？），并且使得destination VM进入replay模式，由此，primary/backup FT结构成功建立起来
- VMotion需要在最后一次switchover发生时，完成所有的未完成的磁盘IO。（什么是switchover？？？？？？？？？？？？？？？）对于primary，这件事情很简单，只需要等待，在物理IO完成（个人猜测是指让宿主IO完成）后将数据传送给primary即可。而backup却很难满足VMotion的要求，因为backup的状态必须与primary一致，而磁盘IO却无法在需要的时间节点完成
    - 解决方法：在最后的switchover节点， backup会通过logging channel要求primary暂时性地完成IO，由于backup会replay primary的操作，因此backup也能完成IO

2.3.5. **容错**

1. 容错的目标：**Output Requirement**。即在primary故障之后，backup接管执行，并且backup的执行与primary已经对外产生的输出是一致的
    - **注意，几乎所有的分布式容错系统均无法保证client只会收到一次数据**。有可能primary在向client发出数据包之后故障，由于backup会replay log entry，因此也会执行同样的操作，进行同样的输出。这时，client就会收到两次数据。但由于backup会完全复制primary相关网络信息，根据TCP协议，在client处，TCP协议可能会认为收到了两个完全一样的重复包，并将第二个数据包直接丢弃
2. primary和backup之间使用UDP heartbeat && 检测logging channel（用于primary给backup发送input）来检测是否存在VM发生了故障。
3. 必须保证即同一时间primary、backups中只有一台机器在进行**实际**执行，即使是在发生了**脑裂（brain split，primary和backups失去了通信）** 的情况下。

- 脑裂的解决：利用共享磁盘。某台VM在go live之前，会对共享数据进行test-and-set操作（原子操作），成功了则可以成为primary，反之则等待。当一台机器因为其他的原因（如磁盘连接等，与primary故障无关）无法访问共享数据时，这一VM可能无法进行工作了。test and set操作可以认为是由磁盘网络上的一个服务器提供的。当这个服务器没有备份并且出现错误时，会导致VMware FT无法选择primary！因此test and set服务器有可能也会进行容错备份

4. $backup\ go\ live$的时间 $\approx$ 故障检测时间 $+$ 执行延迟时间（$execution\ lag\ time$）其中，执行延迟时间即为backup执行log entry中的所有条目，直至与故障时的primary同状态的时间。

5. primary与backup速度不匹配时：

   - 当backup执行速度与primary不匹配（backup过慢）时，会导致backup的状态落后于primary太多，以至于在go live之前需要replay较长时间。同时primary的log buffer容易满，会导致primary暂停。见前文log buffer部分

   - 通常不太会出现backup明显慢于primary的情况，由前文log buffer所述，record和replay的速度基本相同。仅在服务器负荷非常大的情况下才有可能出现backup与primary速度不匹配。

     - 解决办法：在传送log entry以及acks时，会同时传送一些额外的信息用于评估backup的execution lag time，如果这一时间太离谱（超过1s），VMware FT会通知调度器减少给予primary的CPU时间。同时，存在一个反馈循环，用于判断对primary合适的CPU时间限制，以使得primary与backup速度匹配。当backup后续速度增快时，primary的执行速度也会同步增快。

   - **backup的执行也不可以快于primary**：这种情况有可能发生。primary和backup以相近的速度在执行，但是这时primary可能会收到中断，而backup仍然在执行。

     - 解决办法：要求只有在backup的log buffer中至少有一条event的时候才能允许backup执行。因此，backup在刚刚启动时不能执行，必须要等到primary产生第一个logging entry才可以执行（例如定时器中断），backup也会始终落后于primary一个event（log entry）。

6. 当某台backup go live之后，client一定会收到异常信息，而后才能从新的primary获取信息，而不是与旧的primary通信

2.3.5.1. 容错协议

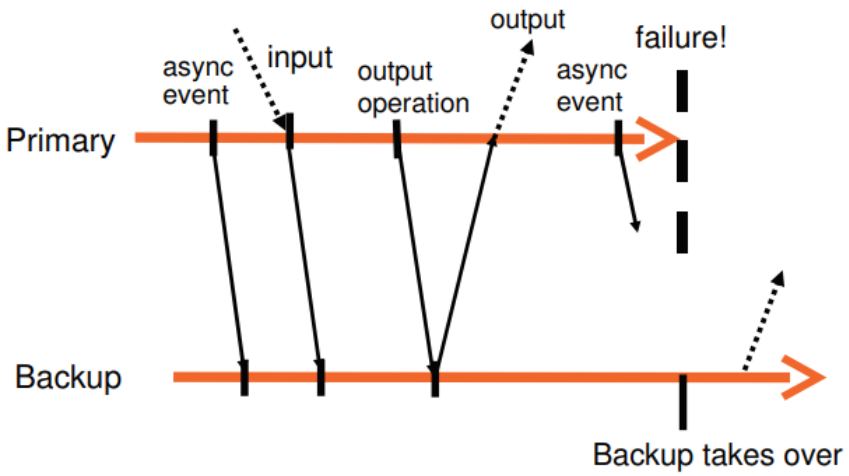


Figure 2: FT Protocol.

1. **Output Rule**：可实现Output Requirement。要求在backup VM在通知primary自己已经收到了 与产生当前输出有关的log entry之前，primary都不能对外传送输出
2. primary并不需要暂停执行，只需要暂停对外输出即可。
3. 因为没有使用**2PC（two phase commit）** 协议，因此无法保证输出只有一次。failover时，backup无法确定primary是在输出之前还是输出之后故障的
4. 网络协议自带对丢包和重复包的处理。对于primary的包可能会丢失，进而导致backup也没收到

2.3.5.2. 故障检测

1. 当backup VM故障时，primary会退出record mode（而后不会再向logging channel发送log entry），正常执行。
2. 当primary故障时，由于backup的执行略有延迟，所以存在一部分log entry没有执行。当log entry中的条目执行完毕后，backup会退出reploy mode，进入normal mode，正式成为primary，这一过程即为**go live**
3. backup failover的过程需要进行一些设备相关的操作
    - VMware FT需要广播新primary的mac地址，让交换机知道新的primary产生
    - 新primary还需要重新进行磁盘IO
4. 故障检测：
    - VMware FT使用UDP心跳来检测故障

- VMware FT还会监控logging channel上的流量。因为现代计算机都会有时钟中断，而中断又会被写入log entry，进而传递给backup，所以logging channel上的流量，**至少**是定时产生的。如果超过一段时间没有在logging channel上检测到流量，则认为存在故障。

2.3.6. 对FT VM的操作

1. 当primary正常关机时，backup也应该关机，而不是go live
2. 当primary相关的系统资源发生变化时，应该也有相应的log entry，并且对backup也做相应的调整
3. 除了VMotion以外，其他所有的操作都应当从primary开始。

2.3.7. 磁盘IO相关问题

1. 磁盘IO之间具有不确定性：
   - 磁盘IO是非阻塞的，可以并发执行。磁盘操作可能访问到磁盘的不同位置
   - VMware使用DMA，而磁盘IO的结果可能会映射到相同的内存块上
   - 解决方案：检测可能发生的磁盘IO竞争。对于那些可能发生竞争的磁盘IO，强制让它们串行执行，并且串行执行的顺序在primary和backup上完全相同
2. 磁盘/网络IO与内存存取之间有不确定性：
   - 同一时间内存可能在读取某个块，同时磁盘操作读取数据，也要写入这个块；其次，IO使用了DMA机制，而DMA的传输需要一定时间，不是一次性的，因此中间会导致不确定性；我们也无法检测何时数据包到达
   - 解决方案一：引入块保护机制。优先保护磁盘操作。若内存中的某个块正在进行磁盘操作，则对这个块进行读写会引发trap。此时，VM会暂停，直到磁盘操作完成了。
   - 解决方案二：改变MMU的保护机制代价太昂贵，引入了 **bounce buffer。即一块暂时的缓冲区，此缓冲区大小与正在进行磁盘操作的内存大小相等** 。读取磁盘数据时，先将数据读取到缓冲区，当IO完成时再将该数据复制到guest内存中；写硬盘数据同样如此，先写入缓冲区，再写入磁盘。bounce buffer是由hypervisor维护的。当IO数据到达时，先将数据写入bounce buffer。当数据完全写入之后，hypervisor会挂起VM，再一次性拷贝到内存之中。而后hypervisor会向guest OS发起模拟的终端，通知数据已经到达。对于backup，随后通过logging channel指明中断发生的指令号，并且传输数据。backup上的VMM同样会挂起backup，一次性传输数据，而后在对应的指令处发起中断通知数据到达

3. 当primary在IO未完成时发生故障时，由于IO请求并不是新的primary发出的（而是已经故障的primary发出的），因此新primary不知道IO是否完成，因此也无法继续执行。由于消除了磁盘IO的竞争，因此磁盘操作是等幂的（idempotent）。所以只需要在backup go live过程中，重新发布磁盘IO的指令即可

2.3.8. 网络IO相关问题

1. 不带FT时，VMware vSphere对虚拟机的网络IO进行了一些异步优化，这些异步优化可能会带来不确定性。对于带FT的情况，相关操作会直接陷入到hypervisor
2. 提升网络IO性能：
    - 方案一：看不懂
    - 方案二：通过避免primary和backup在传输数据时发生线程上下文切换

3. 其他设计

3.1. 非共享虚拟磁盘

1. 论文原始设计中，primary和backup共用同一个虚拟磁盘。磁盘对于primary和backup来说是外部环境，只有primary才能进行相关的磁盘操作，并且需要延迟（直到收到akcs）
2. 若改为非共享磁盘，则虚拟磁盘可以认为是primary和backup的内部环境。如此primary对于磁盘的操作就不需要延迟
3. 当backup故障的时候，primary和backup的磁盘状态可能不一致。因此要在backup重启之后，显式地对两个磁盘的状态进行相关的同步。
4. 非共享虚拟磁盘设计不仅需要同步两部虚拟机的状态（cpu，memory，io等），还需要同步磁盘状态。非共享虚拟磁盘设计主要用于长距离容错。不再有共享数据可以避免脑裂，需要使用其他办法。

3.2. backup读取磁盘

1. 共享虚拟磁盘的情况下，我们将磁盘读取到的数据看做是输入，当primary读取之后，由logging channel发送给backup。若让backup读取磁盘，而非通过logging channel传送，显然可以降低logging channel上的流量。但是会带来一些其他的问题：
    - 降低backup运行的速度。当backup到达某个primary已经完成了的log entry时，它不能向后执行，必须等待磁盘IO完成，否则状态就不再一致

- 当primary读取成功，而backup读取失败时，则backup需要一直重复请求；当primary读取失败时，则需要将对应内存位置的数据通过logging channel发送给backup。
- 当有一个操作序列需要先读取在写入磁盘时，很有可能出现这样的情况：primary读完→backup读→primary写→backup写。在backup读完之前，primary必须要等待，而不能写入磁盘

4. 相关论文：

- HP PA-RISC的容错设计，本文的基础
- fail-stop failure相关
- 同步IO，11
