虚拟机全系统在线迁移

张彬彬[1],罗英伟[1],汪小林[1],王振林[2],孙逸峰[1],陈昊罡[1],许卓群[1],李晓明[1]
(1. 北京大学信息科学技术学院,北京 100871; 2. 美国密歇根理工大学计算机科学系,美国密歇根 49931)

摘 要: 本文介绍了一种虚拟机全系统在线迁移方法. 设计了三阶段迁移方案(Three-Phase Migration,TPM),能迁移包括外存数据在内的虚拟机全系统状态,最小化迁移过程中的虚拟机暂停时间,并保证数据的一致性和完整性. 在该方案中,使用了基于 Block-bitmap 的外存同步方式,由 Block-bitmap 记录迁移过程中的外存更新. 此外,还提出了增量迁移方案(Incremental Migration,IM),当被迁移的虚拟机需要迁移回源主机时,增量迁移能够减少需要迁移的数据量,从而缩短迁移时间,降低迁移造成的性能代价. 通过在 Xen 上的实验表明,该全系统迁移方案即使在 I/O 密集型的负载情况下仍然执行得很好,迁移过程中的虚拟机暂停时间与共享外存迁移时的暂停时间接近. 基于 Block-bitmap 的外存同步机制简单高效,其监控过程中产生的性能代价低于 1 %.

关键词: 虚拟机; 迁移; 全系统; 块位图; Xen
中图分类号: TP312 **文献标识码**: A **文章编号**: 0372-2112 (2009) 04-0894-06

Whole-System Live Migration Mechanism for Virtual Machines

ZHANG Bin-bin[1],LUO Ying-wei[1],WANG Xiao-lin[1],WANG Zhen-lin[2],SUN Yi-feng[1],CHEN Hao-gang[1],XU Zhuo-qun[1],LI Xiao-ming[1]
(1. School of EECS, Peking University, Beijing 100871, China;
2. Department of Computer Science, Michigan Technological University, Houghton, MI 49931, USA)

Abstract: We describe a whole-system live migration mechanism which transfers the whole run-time state including local disk storage, in this paper. A three-phase migration (TPM) algorithm is proposed to minimize the downtime caused by migrating large disk storage data and keep data integrity and consistency. A Block-bitmap is used to track all the write accesses to the local disk storage during the migration, and then to direct synchronization. Also, an incremental migration (IM) algorithm is described to facilitate the migration back to initial source machine. Our experimental study based on Xen shows that the algorithms perform well even when I/O-intensive workloads run in the migrated VM. The downtime of whole-system migration is close to shared-storage migration algorithms. The Block-Bitmap based synchronization mechanism used in both TPM and IM is simple and effective, and the performance overhead of recording all the writes is less than 1 percent.

Key words: virtual machine; migration; whole system; block-bitmap; Xen

1 引言

虚拟机迁移是指将一台主机(源主机)上运行的虚拟机迁移到另一台主机(目的主机)上运行. 为了达到这个目标,需要将虚拟机的运行状态从源主机传输到目的主机,然后在目的主机上恢复虚拟机的运行. 在线迁移是指在整个迁移过程中,虚拟机的暂停时间非常短,虚拟机上运行的服务始终能响应用户的请求[1~3]. 目前大部分虚拟机迁移的研究都只关注源主机和目的主机共享磁盘存储的情况,在这样的情形下,只需要迁移虚拟机的内存和 CPU 状态.

Xen 和 VMware 都实现了共享存储的虚拟机在线迁移,分别称为 Xen live migration[1]和 VMware VMotion[2]. 它们的实现十分类似,以 Xen live migration 为例,它使用预迁移机制,在虚拟机运行的同时,向目的主机循环迁移内存页面,同时记录内存脏页面,每一轮循环仅需要传输上一轮循环过程中产生的脏页面. 当绝大部分内存同步以后,暂停虚拟机,将 CPU 状态和剩余未同步的内存页面同步到目的主机. 然后虚拟机可以在目的主机上恢复运行. 通常情况下,暂停过程中只有少量的内存页面需要同步,暂停时间极短,表现为服务不间断的虚拟机在线迁移. 但是 Xen live migration 和 VMware VMotion 只支持基于共享存储的不包括外存数据的虚拟机迁移.

但是还有一些应用场景,源主机和目的主机之间无

收稿日期:2008-03-26;修回日期:2008-06-04
基金项目:国家 973 重点基础研究发展规划(No. 2007CB310900);国家自然科学基金(No. 90718028,No. 60873052);国家 863 高技术研究发展计划(No. 2008AA01Z112);教育部-英特尔信息技术专项科研基金(No. MOE-INTEL-08-09);华为科技基金(No. YJCB2007002SS)

共享存储设备,当这些机器上的虚拟机需要迁移时,本地磁盘数据也必须传输到目的主机.这种包括外存数据在内的虚拟机迁移称为全系统迁移.

外存迁移需要同步的数据量大,是全系统迁移的关键,目前有三种外存迁移方式:

一是停机迁移(freeze-and-copy).该方式在虚拟机暂停过程中迁移所有状态数据,因此带来极大的暂停时间.使用该方式的著名系统包括 Internet Suspend/Resume (ISR)[4,5]和 Collective[6,7],主要面向无实时性要求的应用,比如可以使用 ISR 来实现个人移动计算环境:通过一个第三方的网络存储中转数据,当用户在 A 地使用完虚拟计算机时,将其暂停(Suspend)并存储成一个虚拟机映像文件,保存到网络存储中.当用户移动到 B 地时,从网络存储获得虚拟机映像文件,就可以恢复(Resume)虚拟机运行.为了缩短停机时间,Collective 项目也提出了一些改进措施以减小需要传输的数据量,其中一项改进是使用写时复制(Copy-on-Write)技术记录磁盘的所有更新,从而只需要迁移最近写入的数据,在一定程度上实现了增量迁移.

二是按需取页(on-demand fetching)[5].该方式先迁移虚拟机的内存数据和 CPU 状态,当虚拟机在目的主机上恢复运行后,再根据读写磁盘请求,从源主机取数据.该方式可以达到与基于共享存储的在线迁移相同的暂停时间.但是会造成对源主机长时间的依赖.因此这种迁移无法用于源主机的维护、平衡负载等需要关闭源主机或降低源主机负载的应用场景,也不适用于地理位置分布的计算平台.此外,该方式使被迁移的虚拟机长期同时依赖于两台物理主机,将降低该虚拟机的可用性.

第三种方式是德国的 Robert Bradford 等提出的预迁移结合基于回放的同步方式实现的外存迁移[8].首先预迁移虚拟机的外存数据,在此过程中截获所有写外存的请求,包括写数据、写的位置及数据长度,并同时转发到目的主机按序保存.当外存预迁移结束后,目的主机将重做这些写操作.在内存预迁移过程中发生的写操作同样记录到目的主机的队列中.当虚拟机在目的主机恢复运行后,先阻塞磁盘 I/O,直到重做完队列中的写操作.这种方案暂停时间短,但是对于写密集型应用,虚拟机在目的主机恢复以后,可能有较长的磁盘 I/O 阻塞时间.此外,由于写操作的空间局部性,所记录的写操作序列中可能存在一定数量的冗余记录.

理想的全系统在线迁移应该最小化暂停时间,尽可能降低迁移总时间和迁移数据量,并降低性能代价.此外还应该保证迁移过程不会造成对源主机的长期依赖,并且保证迁移过程对客户操作系统透明.

针对这些目标,基于现有研究,我们提出了一种三阶段迁移方案(Three-Phase Migration,TPM),该方案包括预迁移、停机迁移、后续迁移三个阶段.与 Robert Bradford 等的记录和转发并重放写操作的同步方案不同,本文采用 Bitmap 记录外存数据是否需要同步,性能开销较低,避免了记录的冗余,避免了 I/O 阻塞,实验表明,该方式在 I/O 密集的应用场景也能有效工作.

基于三阶段迁移,我们还提出了增量迁移方案,该方案只针对虚拟机需要在源主机和目的主机之间往复迁移的应用场景,在迁移时只需要同步最近更新的数据,可以降低迁移时间和迁移数据量.

2　三阶段迁移

2.1　设计

在线迁移要求该同步过程中只能有极短的虚拟机暂停时间,而全系统迁移则需要同步大量的数据,因此如何解决较短的暂停时间和同步大量数据的矛盾,是全系统在线迁移面临的一个关键问题.

我们采用的三阶段迁移(TPM),其流程如图 1 所示.

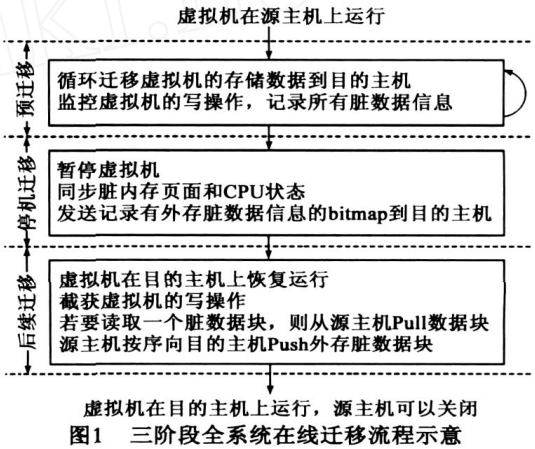

图1　三阶段全系统在线迁移流程示意

(1)预迁移阶段,本地存储数据循环地传送到目的主机,在此过程中记录所有写操作作用的数据的位置.第一轮循环传输所有存储数据,以后的逐轮循环只需要根据上一轮循环的记录传输上一轮循环中被修改过的数据.当循环次数到达最大限度时,或脏数据产生的速率大于传输速率时,预迁移结束.

(2)停机迁移阶段,虚拟机被暂停,同步内存脏页面和 CPU 状态;源主机向目的主机发送所有未同步的外存数据块的位置信息.

(3)后续迁移阶段,虚拟机在目的主机上恢复运行.此时,除部分外存数据外,其他状态在源主机和目的主机之间已经完全同步.采用 Push 和 Pull 结合的策略完成外存数据的同步.当虚拟机试图读取脏数据块时,它主动从源主机 Pull(这会导致一定程度的延时);而源主机则顺序地向目的主机 Push 脏数据块以保证数

据能够在有限的时间内完成同步,同时保证优先传送被 pull 的数据块.

三阶段迁移的关键是外存数据的同步,并且内存迁移已经有较好的研究基础,以下只讨论外存数据的同步.

2.1.1 Block-Bitmap

我们采用一个 Bitmap 来记录迁移过程中的脏数据块的位置. Bitmap 中的每一位(Bit)对应一个外存单元,该位为 1 表示该单元不一致,需要同步. 其设计主要考虑以下两个问题:

监控粒度:指 Bitmap 的每一位可描述的外存数据单元大小. 该数据单元必须满足原子特性,即不存在对该单元某一部分的 I/O 操作. 在文件系统层次,外存操作的基本单元是 4kB 大小的块(Block),我们选择 4kB 块作为监控单元,把该 Bitmap 称为 Block-bitmap. 一个 32GB 的磁盘只需要 1MB 的 Block-Bitmap.

代价:使用 Block-bitmap 记录磁盘脏数据块产生以下代价:一是在内存中维护 Block-bitmap 的空间代价,二是遍历 Block-bitmap 的时间代价. 根据磁盘写操作的局部性特点,我们设计了一个层次化的 Block-bitmap,将 Block-bitmap 划分成若干部分,各部分的状态由一个上层的 Bitmap 描述,遍历时,仅当上层的某一位是 1 时,才检查对应的下层区段. 此外,下层的区段只在第一次写该段时分配空间.

虽然层次化 Block-bitmap 可以减少外存迁移的控制时间,但实验表明,若虚拟机的外存不大,Block-bitmap 带来的开销相较于全系统迁移代价可以忽略.

2.1.2 外存数据同步

在基于 Block-bitmap 的外存数据同步方式中,所有不一致数据由 Block-bitmap 标记,所有同步操作依据 Block-bitmap 执行.

在预迁移阶段,当每一轮迁移开始时,Block-bitmap 拷贝一个副本,迁移进程根据副本,把其中标记为 1 的数据块传输到目的主机,而原 Block-bitmap 被清空以标记新一轮预迁移过程中的脏数据块.

在暂停迁移阶段,源主机向目的主机传递一个记录了最后一轮预迁移后所有待同步脏数据块的 Block-bitmap 副本.

在后续迁移阶段,当被迁移的虚拟机恢复运行时,目的主机要先创建一个新的 Block-bitmap,用于标记被迁移虚拟机在目的主机上的所有更新,该 Block-bitmap 将被用于第 3 章中介绍的增量迁移. 该阶段,目的主机向源主机 Pull 读操作访问的脏数据块;同时源主机将 Block-bitmap 中标记的脏块 Push 到目的主机,以保证迁移在有限时间完成;源主机优先发送被请求的数据. 目的主机的同步流程如程序段 1 所示.

程序段 1 后续迁移阶段目的主机的同步流程

```
定义
—I/O 请求 R〈O,N,VM〉,其中 O 表示操作类型,为 WRITE 或 READ;
  N 表示外存数据块号,VM 是提交该请求的虚拟机的 ID
—transferred_block_bitmap:在暂停阶段由源主机传送给目的主机,
  标记了所有源主机和目的主机间不一致数据块
—new_block_bitmap:标记被迁移虚拟机在目的主机上发生的所有
  脏数据块
1. An I/O request R〈O,N,VM〉is intercepted;
2. Queue R in a pending list P;
3. IF R.VM! = migrated VM
4.       THEN goto 14;
5. IF R.O = = WRITE
6.       THEN{
7.          new_block_bitmap[N] = 1;
8.          transferred_block_bitmap[N] = 0;
9.          goto 14;
10.      }
11. IF transferred_block-bitmap[N] = = 0//此时 R.O = = READ 且读
    取的数据块非脏
12.      THEN goto 14;
13. Send a pulling request to the source machine for block N,goto 16;//读
    取脏数据块
14. Remove R from P;
15. Submit R to the physical driver;
16. End.
```

同时,目的主机要分别处理接收到的 Pull 和 Push 的数据块,如程序段 2 所示.

程序段 2 目的主机对接收的同步数据的处理

```
1. A block M is received;
2. IF transferred_block_bitmap[M] = = 0//该数据块已在目的主机端
   写脏,丢弃即可
3.          THEN goto 12;
4. Update block M in the local disk;
5. transferred_block_bitmap[M] = 0;
6. FOR each request Ri in P//判断本次数据块是否是 Pull 过来的
7.       IF Ri.N = = M
8.          THEN{
9.          Remove Ri from P;
10.          Submit Ri;
11.          }
12. End.
```

2.2 TPM 实现

我们对 Xen 的在线迁移进行了扩展,实现了一个 TPM 的原型.

我们在 Xen 在线迁移前实现外存预迁移,因为内存数据的更新率远高于外存数据,并且外存数据的预迁移持续时间较长,在外存数据迁移过程中会产生大量的内存脏数据. 在 Xen 在线迁移的虚拟机暂停阶段

完成 Block-bitmap 的传输. 增加了一个用户进程完成外存预迁移和后续迁移.

修改了 Xen 的后端驱动(Backend driver),使其截获被迁移虚拟机的所有磁盘 I/O 写请求,并将被更新的脏块标记在一个 Block-bitmap 中.

3　增量迁移

三阶段迁移(TPM)虽然能控制虚拟机暂停时间,并保证迁移过程能够在有限时间内完成,不会造成对源主机的无限依赖,但由于外存数据量极大,迁移时间仍然较长.

若迁移后在目的主机保持记录磁盘的脏数据信息,则当虚拟机需要迁回源主机时,只需要同步修改过的数据块,这将大大减少迁移数据量从而缩短迁移时间. 据此,我们给出了增量迁移(Incremental Migration,IM)方案,当虚拟机的迁移来回发生在两个主机之间,则增量迁移能极大提高迁移的效率. 在实际应用中,往复迁移的例子很多,如:需要维护主机时,将虚拟机迁到一台备份主机,维护完成后,需要将虚拟机迁回;或者远程办公,需要将虚拟机在办公室和家之间来回迁移.

由于内存数据变化快,且制约全系统迁移时间的主要因素是比内存大很多的外存数据量,因此在全系统增量迁移的设计中,只考虑外存的增量迁移.

当虚拟机在目的主机上运行时,继续采用 Block-bitmap 方法标记修改的外存数据块,当需要将虚拟机迁回源主机时,检查 Block-bitmap,仅传输其中标记为脏的数据块. 简单修改前述三阶段迁移就可以实现增量迁移:当虚拟机在目的主机上启动时,创建一个 Block-bitmap 标记在目的主机上被更新的数据块. 当需要将虚拟机迁回源主机时,该 Block-bitmap 记录了此时两个主机上不一致的数据块. 此时进行外存数据的第一轮预迁移只需根据该 Block-bitmap 进行即可,其余过程则无需修改. 需要说明的是,回迁过程中也创建新的 Block-bitmap 记录迁移过程中磁盘的变化,这有别于前面用于记录虚拟机迁移前磁盘变化的 Block-bitmap.

若是第一次迁移,则在预迁移前为虚拟机初始化一个全为脏的 Block-bitmap,保证迁移所有数据.

4　评估

本节测试评估三阶段迁移和增量迁移的性能,包括暂停时间、总迁移时间、迁移数据量、性能代价等.

4.1　实验环境

实验中,我们采用了三台机器. 其中两台机器作为虚拟机的主机,硬件配置为 Core 2 Duo 6320 CPU、2GB 内存、SATA2 硬盘. 软件环境为 Xen-3.0.3,虚拟机上运行 XenoLinux-2.6.16.29. 每台主机运行两个虚拟机,一个是被迁移的虚拟机,使用 512MB 内存和 4G 虚拟磁盘,另一个是特权虚拟机(Domain0),使用剩余的机器内存. 为避免虚拟机间切换对性能的影响,两个虚拟机分别使用不同的 CPU 核. 第三台机器作为客户端,访问被迁移的虚拟机运行的服务. 三台机器使用千兆网连接.

实验时,将非特权虚拟机由一台主机迁移到另一台主机,以评价三阶段迁移的性能. 当虚拟机在目的主机上运行一段时间后,再迁回源主机,以评价增量迁移的性能.

4.2　工作负载

全系统迁移的关键问题在于外存迁移,所以我们选择了一组不同 I/O 负载的应用运行在被迁移的虚拟机上,一方面考察不同的 I/O 负载对迁移的影响,另一方面考察迁移对 I/O 性能的影响. 这组应用包括:(1) 动态 Web 应用服务器,在运行过程中可能会产生突发的大量读写操作;(2) 视频服务器,在运行过程中产生持续均衡的读操作及少量用于日志的写操作,对延迟敏感;(3) 一个用于衡量 I/O 性能的基准测试程序,能够持续产生大量读写操作. 这些程序常用于虚拟机迁移性能评测.

4.3　实验结果

所有实验中,在客户端看来,被迁移虚拟机上的服务始终保持运行.

表 1　被迁移虚拟机上运行不同工作负载时的迁移性能指标

	动态 Web	视频服务	I/O 密集型
总迁移时间/ms	61315	61487	108701
暂停时间/ms	52	44	57
传输数据量/MB	4010.7	4009.5	5227.4

从表 1 可以看出,该原型保证了较短的暂停时间,与 Xen 的在线迁移接近;迁移能在有限时间内完成;迁移过程中传输的数据量与虚拟机使用的磁盘大小接近,表明基于 Block-bitmap,用于同步的额外数据较少.

4.3.1　动态 Web 服务器测试

我们在待迁移的虚拟机上配置 SPECweb2005 (http://www.spec.org/web2005/) 服务器,运行 Banking Server 服务,并配置 50 个并发的 Session 作为工作负载.

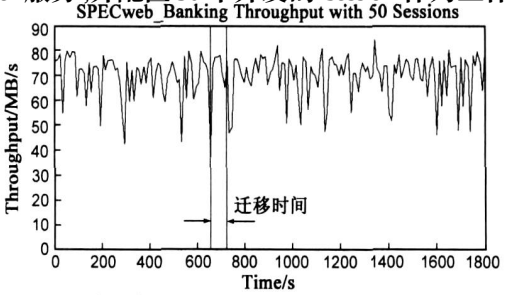

图2　迁移过程中SPECweb2005 Banking Server的吞吐率

如图 2 所示,在迁移过程中,没有产生明显的吞吐率下降.

该实验中,预迁移阶段进行了三轮外存数据传输,共重传了 530 个数据块.后续迁移阶段持续了 74 毫秒,同步了 107 个数据块,其中只有一个数据块是由目的主机 Pull 的,几乎不会对虚拟机的读写产生影响.

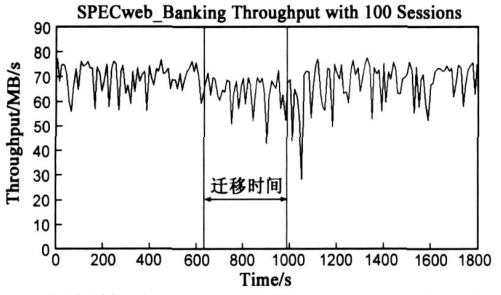

图3 负载增加时SPECweb2005 Banking Server的吞吐率

当负载增加到 100 个并发 Session 时,吞吐率的变化情况如图 3 所示,在迁移过程中吞吐率有所降低,迁移时间也变长,其原因在于大部分网络带宽被用于 SPECweb Banking Server 服务.在这种场景下,较合理的配置是使用一块单独的网卡完成迁移.

4.3.2 低延迟服务测试

我们把待迁移虚拟机配置为 Samba 服务器(http://us1.samba.org/samba/),向一个 Windows 的客户端共享一个视频文件(rmvb 格式).在客户端播放该视频时,迁移该虚拟机.在整个迁移过程中,视频播放流畅.由于该测试中虚拟机上写操作很少,在预迁移阶段只进行了两轮外存复制,仅重传了 60 个数据块;后续迁移持续了 48 毫秒,同步了 4 个数据块.

4.3.3 高 I/O 负载服务测试

在待迁移虚拟机上运行 I/O 基准程序 Bonnie++(http://www.coker.com.au/bonnie++/),然后迁移该虚拟机,观察迁移过程中 Bonnie++ 的性能.

Bonnie++ 以较高的速率写磁盘,因此在迁移过程中,有较多的数据块被更新.在预迁移阶段,约有 804MB 数据被重传,该过程持续了 104 秒,导致了较长的迁移总时间.但是 Block-bitmap 大小固定,因此,不会对暂停时间产生额外影响.

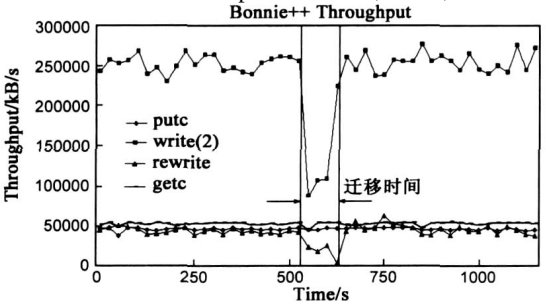

图4 迁移过程中I/O性能的变化

如图 4 所示,迁移过程中,虚拟机的 I/O 性能明显下降,这是因为迁移进程的大量读磁盘操作.

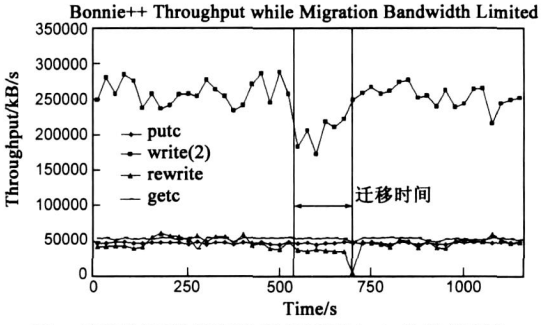

图5 当限制迁移带宽时 迁移过程中I/O性能的变化

若限制迁移使用的带宽,则可以降低迁移造成的磁盘 I/O 负载,将较明显的改善了迁移过程中Bonnie++ 的性能,如图 5,但迁移总时间相应变长.

4.3.4 增量迁移测试

虚拟机迁移到目的主机以后,运行一段时间,再将其迁回源主机,以测试 IM 性能.本组对比测试中,被迁移的虚拟机使用 40G 的虚拟磁盘.结果如表 2 所示.

表 2 首次 TPM 迁移与 IM 迁移的比较(时间:s,数据量:MB)

	动态 Web 服务		视频服务		I/O 密集应用	
	迁移时间	传输数据量	迁移时间	传输数据量	迁移时间	传输数据量
首次 TPM	796.1	39097	798.0	39072	957	40934
IM	1.0	52.5	0.6	5.5	17	911.4

从表 2 可以看出,增量迁移中传输的数据总量远小于首次三阶段迁移传输的数据总量,因此总迁移时间也相应的极大减少.

4.3.5 基于 Block-bitmap 的同步机制带来的 I/O 性能代价

我们将虚拟机运行在 I/O 操作被监控的状态,即拦截所有写操作,并在 Block-bitmap 中标记被更新的数据块,这与迁移过程中的情况是一致的,我们在这样的虚拟机上运行 Bonnie++,其性能与正常的虚拟机上运行的 Bonnie++ 的性能相比较,结果如表 3 所示.

表 3 同步机制带来的 I/O 性能开销

单位:kB/s	putc	write(2)	Rewrite
无写监控	47740	96122	26125
有写监控	47604	95569	25887

从表 3 可以看出,基于 Block-bitmap 的同步机制带来的 I/O 性能开销小于 1%.

5 结论和展望

本文介绍了三阶段全系统在线迁移(TPM)方案,该

方案使用基于 Block-bitmap 的方式同步外存数据,保证了迁移过程中较短的虚拟机暂停时间,并避免对源主机的长时间依赖.进一步提出了增量迁移(IM)方案,该方案利用 Block-bitmap 记录虚拟机在目的主机运行时的外存更新,当虚拟机回迁时,只需迁移更新过的数据块,极大减小了需要迁移的数据量,缩短了迁移总时间.

本文所设计的 TPM 和 IM 算法将虚拟机看作是一个黑盒子,虚拟磁盘的所有数据都将被迁移,即便其中有一些块并未使用.若 Guest OS 能够通知迁移进程磁盘的使用情况,则迁移数据量将大大减少.另一方面,若从 Guest OS 安装时就由 Block-bitmap 记录虚拟机使用的磁盘块;或者按需分配虚拟机的外存空间,则可以只迁移实际使用的外存数据.这些策略需要进一步实验验证.

此外,本文的 IM 只能用于前一次迁移的源主机和目的主机之间.进一步,我们希望能实现虚拟机的外存数据版本管理,从而能够在多台物理主机间实现增量迁移.

参考文献:

[1] C Clark, K Fraser, S Hand, J G Hansen, E Jul, C Limpach, I Pratt, A Warfield. Live migration of virtual machines[A]. In Proceedings of the 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation(NSDI 2005)[C]. Berkeley, CA, USA: USENIX Association, 2005. 273 - 286.

[2] M Nelson, B Lim, G Hutchins. Fast transparent migration for virtual machines[A]. In Proceedings of the 2005 USENIX Annual Technical Conference[C]. VMware Press, 2005. 391 - 394.

[3] J G Hansen, E Jul. Self-migration of operating systems[A]. In Proceedings of the 11th ACM European SIGOPS Workshop[C]. New York, NY, USA: ACM, 2004. 126 - 130.

[4] Kozuch M, Satyanarayanan M. Internet suspend / resume[A]. In Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications[C]. Washington, DC, USA: IEEE Computer Society, 2002. 40 - 46.

[5] M Kozuch, M Satyanarayanan, T Bressoud, C Helfrich, S Sinnamohideen. Seamless mobile computing on fixed infrastructure[J]. IEEE Computer, 2004, 32(7): 65 - 72.

[6] C P Sapuntzakis, R Chandra, B Pfaff, J Chow, M S Lam, M Rosenblum. Optimizing the migration of virtual computers[A]. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation(OSDI 2002)[C]. New York, NY, USA: ACM, 2002, Volume 36: 377 - 390.

[7] R Chandra, N Zeldovich, C Sapuntzakis, MS Lam. The collective: a Cache-based system management architecture[A]. In Proceedings of the Second USENIX/ACM Symposium on Networked Systems Design and Implementation(NSDI 2005)[C]. Boston, MA, 2005. 259 - 272.

[8] R Bradford, E Kotsovinos, A Feldmann, H Schioberg. Live wide-area migration of virtual machines including local persistent state[A]. In Proceedings of the 3rd International Conference on Virtual Execution Environments(VEE '07)[C]. New York, NY, USA: ACM, 2007. 169 - 179.

作者简介:

张彬彬　女,1982 年 1 月出生于云南大理.2004 年获北京大学理学学士学位,现为硕博连读生,从事系统虚拟化方面的有关研究.
E-mail: zhangbinbin@geoagent.pku.edu.cn

罗英伟　男,1971 年 10 月出生于江西吉安,博士,副教授,中国计算机学会高级会员,系统软件和体系结构专委会委员.1999 年在北京大学获得理学博士学位.主要研究方向为系统虚拟化、地理信息系统.(本文通讯作者)
E-mail: lyw@pku.edu.cn