虚拟机磁盘迁移技术研究与实现

吕小虎 李 沁
(北京航空航天大学计算机学院 北京 100191)

摘 要 已有基于内存的虚拟机迁移技术要求迁移源机和目标机之间必须共享网络磁盘,迁移性能受网络条件的影响很大,且在不支持"共享网络磁盘"的环境中,无法实现虚拟机迁移。针对上述问题,考虑到虚拟机磁盘作为虚拟机运行所需的持久化状态的封装体,所展现出的如高可用性、高效能、安全稳定等良好特性,提出基于磁盘的虚拟机迁移。通过设计虚拟磁盘驱动系统DiskMig,在实现虚拟机磁盘迁移的同时保证了其在源端和目的端的一致性。设计了"Transfer on Demand with Forward-ahead"(TOD&FA)算法快速同步源端和目的端磁盘文件差异。DiskMig采用自行设计的高效存储结构bitmap,使记录、查询虚拟机磁盘迁移过程中的大量I/O操作时仅需时间复杂度O(1)。实验表明,DiskMig所采用的相关方法对虚拟机及其中应用程序运行性能的影响仅为5%左右,有效支持了虚拟机磁盘文件的迁移。

关键词 虚拟机迁移,虚拟机,一致性,虚拟磁盘驱动
中图法分类号 TP302.1 **文献标识码** A

Research and Implementation on Migration of Virtual Machine Including Vm-disk

LU Xiao-hu LI Qin
(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract The existing memory-based virtual machine(VM) migration technology requires network-disk shared between source side and destination side, which results performance degradation greatly from poor network quality, and the VM migration can not be implemented without shared network-disk environment. To solve these problems, in view of the high-availability, high-performance, security and stability displayed by VM-disk, which acts as the encapsulation of VM persistent running state, a new migration strategy based on VM disk was proposed and implemented. A virtual disk driver named DiskMig was designed to migrate VM disk and ensure its consistency during migration process, and a new algorithm Transfer on Demand with Forward-ahead(TOD&FA) was presented to synchronize disk differences between source and destination fast. An effective storage structure bitmap was adopted by DiskMig system to record and search massive I/O offset information during migration with only constant time complexity. The experiment demonstrates that the performance of VM and its applications brought by the DiskMig system was only 5 percent degraded, thus the VM migration including its disk is implemented efficiently.

Keywords Virtual machine migrate, Virtual machine, Consistency, Virtual disk driver

虚拟机技术所具有的透明性、隔离性、封装性等特点可有效屏蔽硬件平台的异构性、实现软件运行的隔离和硬件资源的共享。因此虚拟机技术成为一个热点研究问题,如斯坦福大学的VMware、剑桥大学的Xen项目[1]等开展了卓有成效的研究。作为虚拟机技术的重要应用,虚拟机迁移技术是将某个OS及其上运行的所有应用作为一个完整封装体迁移,保证软件在硬件设备升级、维护、失效时可靠、透明地运行,实现计算系统动态负载均衡,有效避免了传统进程迁移中存在的"遗留应用"[2]瓶颈问题。

目前已有的具有代表性的迁移技术只涉及迁移虚拟机内存状态,迁移源机和目标机之间必须通过共享网络磁盘的方式实现对磁盘内容的访问。这种方式中,迁移性能受网络条件的影响很大,在不具备"共享网络磁盘"的条件下,则无法实现虚拟机迁移;而已有的带磁盘的迁移方式要求迁移完成后目的端虚拟机必须禁止一切I/O操作,以同步迁移过程中所产生的磁盘差异,导致I/O利用率低,虚拟机中应用程序执行耗时增加。

本文提出基于虚拟机磁盘的迁移,基于Linux内核设计了DiskMig系统,以控制迁移过程前后虚拟机磁盘文件的I/O行为及一致性问题;实现了bitmap存储结构,以高效解释并记录迁移过程中源端虚拟机大量I/O变化信息;设计了"Transfer on Demand & Forward-ahead"同步算法,根据迁移后虚拟机在目的端运行的动态需要,并通过预测程序执行过程中I/O行为,快速同步磁盘差异。

到稿日期:2008-08-12 返修日期:2008-09-11 本文受国家973项目(编号:2005CB321803),国家863项目(编号:2007AA01Z426)资助。
吕小虎(1983-),男,硕士生,主要研究方向为虚拟计算等,E-mail:lvxiaohu@act.buaa.edu.cn;李 沁(1982-),男,博士生,主要研究方向为虚拟计算等。

1 相关研究工作

目前，虚拟机迁移分为局域网环境下迁移和广域网环境下迁移两大类。局域网环境下的研究只需迁移虚拟机运行时的内存状态。最简单的方式是“停机拷贝”方式，完全停止虚拟机运行，将其内存状态从源端迁移到目的端。该方式虽然简单，但必须停止虚拟机及其中应用的运行，需较长的应用程序中断时间(down-time)，迁移过程对用户很不透明，迁移效率较低[10]。直到 2004 年，VMware 和 Xen 项目先后分别提出了 Vmotion[3]和“Xen Live Migration”方案[4]，提出了保持虚拟机及其中应用运行的同时迁移虚拟机，即在线迁移概念[11]。以 Xen 为例，采用在源端保持虚拟机运行时“预传输”的方式迭代 N 轮，预传输其内存状态到目的端，只需在最后一轮迭代中停止虚拟机运行；取得较高迁移效率：down-time 只需 100ms。该方式的缺陷在于，不支持传输虚拟机磁盘文件，在迁移过程中迁移源机和目标机必须共享网络磁盘(NAS 方式)[4]。该共享方式只在局域网环境下表现出较好的可用性，无法在非共享网络磁盘的宿主机之间迁移，应用范围有限，难以聚合分布性较强的资源。此外，该方式对网络性能要求苛刻，如网络环境较差，导致迁移性能下降，进而影响了虚拟机中应用的运行；而许多实际应用，如 database-server 和 dynamic web-server，需要将其运行状态保存在本地存储中，而不是频繁地访问远端的共享文件服务器。

考虑到虚拟机磁盘作为虚拟机运行所需的持久化状态的封装体，所展现出的如高可用性、高效能、安全稳定等良好特性，因此基于虚拟机磁盘的迁移逐渐成为虚拟机迁移(尤其是广域网络环境下)的研究热点[4,5,9]。该类迁移概括为 3 种方案：①Freeze-and-copy[6]方案停止虚拟机运行，先传输其磁盘文件，随后在目的端启动虚拟机，虽然简单，但是等待时间难以忍受，效率很低；②保持虚拟机运行的同时传输其磁盘镜像，然后在目的端完全停止虚拟机运行，利用 rsync 程序同步磁盘差异，该方案虽然有一定的进步性，但虚拟机停止时间仍然较长；③Robert Braford 等人提出的广域网下类似于“Xen Live Migration”的迁移方案[7]，首先磁盘预迁移文件到目的端，然后开始 Xen 内存迁移，在这两个阶段都需要记录磁盘变化，通过扩充 Xen 监控器中特有的 back-end driver 代码功能，提供了记录源端虚拟机磁盘文件在虚拟机迁移过程中发生的写变化的模块，最后在目的端恢复虚拟机运行并恢复源端系统的磁盘差异。该方案的特点是：广域网环境下的 Live Migration、较好的停机时间(68s 左右)、保持了迁移过程的透明性和一致性。其不足在于：虚拟机在目的端恢复运行时，禁止一切 I/O 操作，须等到将源端磁盘写变化全部恢复到目的端后才能启动 I/O 操作，产生“I/O 饥饿”现象，导致虚拟机中程序的 task-time 增加，I/O 利用率低。

2 虚拟机磁盘迁移问题分解

2.1 迁移过程的透明性

在迁移过程中，由于虚拟机中许多应用程序具有实时性较高、与用户交互性较强的特点，因此要求迁移过程具备“在线迁移”特性，即将正在运行的虚拟机及其中的应用程序几乎不间断地从一台物理机器迁移到另一台机器上，从而可以在对上层软件完全透明的情况下，保证软件在硬件设备升级维护、失效时不间断运行。

考虑到目前虚拟机所存在的上百 MB 的内存及大容量的磁盘，若只是通过简单停止虚拟机运行来传输这些数据，显然导致停机时间过长，无法实现“在线迁移”的要求。因此本文采取“单轮预拷贝磁盘”的方法来实现“在线迁移”，即保持虚拟机在源机运行的同时由后台把磁盘文件透明传输到目的端，传输完成后只需在目的端同步不一致的磁盘数据，以尽可能减小因磁盘传输给虚拟机及其应用带来的停机时间。

2.2 虚拟机磁盘文件一致性问题

由于“单轮预拷贝磁盘”过程中，虚拟机仍在源机保持运行，产生许多新的 I/O 变化信息，因此要保持虚拟机在迁移前后运行状态的连续性和一致性。首先需要保证磁盘状态在源端和目的端的一致性，如何在迁移前后保持磁盘状态一致性，是本文研究的关键问题。通过分析，迁移过程中磁盘文件一致性问题可分解为如下两种场景。

(1)虚拟机在源端运行

1)disk I/O = ‘read’ 读操作不会改变磁盘状态，因此对于磁盘读操作正常进行。

2)disk I/O = ‘write’ 引入二元组 $V=\langle T,D\rangle$，其中 T 表示时间，D 表示磁盘状态集，另设 W 表示所有对磁盘的写操作集。设虚拟机在$[T_0,T_s]$运行于源端，$[T_{p1},T_{p2}]$为‘磁盘预迁移’阶段($[T_{p1},T_{p2}]\subseteq[T_0,T_s]$)，$t_d$ 为虚拟机在目的端起始运行时刻($T_d\geqslant T_s>T_0$)；设 t_1 时刻($t_1\in[T_{p1},T_{p2}]$) $v=\langle t_1,d_1\rangle(v\in V)$，此时在 D 集发生 w 操作($w\in W$)，表示为 $d\xrightarrow{w}d'$，则 $v=\langle t_1,d'\rangle$；又设在 t_2 时刻($t_2>T_d$)虚拟机在目的端，读取 $v=\langle t_1,d_1\rangle$的内容，显然有$\langle t_1,d'\rangle\neq\langle t_1,d_1\rangle$。因此得出：虚拟机在源端运行时，需对‘写’操作进行控制。

(2)虚拟机在目的端运行

1)disk I/O = ‘read’ 同(1)中 2)部分论证。

2)disk I/O = ‘write’ 各状态变量同(1)，设 t_2 时刻虚拟机在目的端($t_2>T_d$)，$v_1=\langle t_1,d_1\rangle(t_1\in[T_{p1},T_{p2}])$，此刻在 D 集执行 w 操作($w\in W$)，表示为 $d_1\xrightarrow{w}d_2$，则 $v_2=\langle t_2,d'\rangle$。如果即使 $\exists t_x$ 时刻($t_1<t_x<t_2$)对 D 进行读取操作，则同(1)中 2)部分；如 $\exists t_x$ 时刻($t_1<t_x<t_2$)对 D 执行 w 操作($w\in W$)：$d_1\xrightarrow{w}d_x$，$v_x=\langle t_x,d_x\rangle$；而 $d_1\xrightarrow{w}d_x\xrightarrow{w}d_2\Leftrightarrow d_1\xrightarrow{w}d_2$，因此，得出：虚拟机在目的端运行时，只需对‘读’操作进行控制。

为此，本文设计了基于 Linux 内核具有 3 种迁移模式的 DiskMig 系统，以控制虚拟机对其磁盘的 I/O 操作及保证迁移过程中磁盘状态的一致性。

2.3 尽可能短的服务执行时间和中断时间

前面所述“在线迁移”技术只能达到减少虚拟机及其中应用程序的停机时间的目的。而实际迁移过程中，如何获得尽可能短的程序中断时间及程序总体执行时间，也是影响迁移性能的关键。对迁移过程中应用程序运行总耗时分析如下。

设 T 表示某应用 App 在迁移存在的情况下的总体运行

耗时，T_s 表示在“预拷贝磁盘”阶段该应用运行时间，T_d 表示虚拟机迁移到目的端后该应用运行所经历的时间，T_{sync} 表示因同步磁盘差异而引发的程序等待时间，因此存在 $T=T_s+T_d+T_{sync}$。由此可得：欲获得程序总体执行时间 T，必须使同步时间 T_{sync} 尽可能小，而同步时间 T_{sync} 则取决于如下两个因素：

1)所需同步的磁盘差异量；

2)磁盘差异同步方法。

为实现上述目标，本文设计 bitmap 结构，高效存储迁移过程中虚拟机磁盘 I/O 变化信息；当虚拟机迁移到目的端继续运行时，采用“Transfer on Demand & Forward-ahead”算法快速同步磁盘差异等方式，以提高迁移性能。

3 虚拟机磁盘迁移流程

基于上述问题描述，给出虚拟机磁盘迁移流程如图 1 所示。

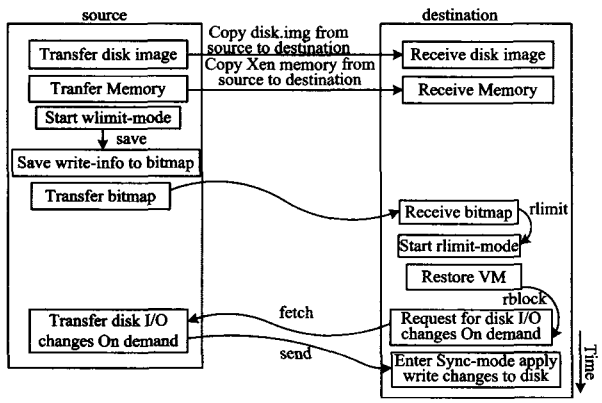


图 1 系统流程图

1)保持虚拟机在源机运行的同时，后台把磁盘文件透明传输到目的端。

2)基于 Xen 虚拟机控制器平台，调用其内存迁移功能，将虚拟机内存状态从源端迁移到目的端。

在上述两步中，由于源端虚拟机运行造成磁盘状态变化，故需把磁盘 I/O 变化记录到自行设计的 bitmap 结构中。

3)传输源机把磁盘变化位置(bitmap 结构)记录到目的端。

4)在目的端恢复虚拟机继续运行。在保证其 I/O 操作继续进行的前提下，如果某 T 时刻磁盘上虚拟机运行所需某偏移位置 pos 处的数据存在差异，依据“TOD”策略，动态通知源端按需传输该部分数据，并由源端通过预测不同应用程序 I/O 分布特性，由该位置起把若干偏移位置对应的数据预传输到目的端。

4 DiskMig——虚拟机磁盘迁移系统

4.1 DiskMig 3 种迁移模式

基于对磁盘一致性分析，本文通过扩充 linux 系统的虚拟磁盘驱动 loop，设计具有 3 种迁移模式的 DiskMig 系统。DiskMig 将迁移过程中虚拟机磁盘 I/O 操作抽象为 write-limit，read-limit 和 Sync 3 种模式，以支持迁移虚拟机磁盘，控制迁移过程中磁盘状态的一致性。之所以未采用传统的基于 Xen split-driver 结构的方式[8]，是由于该方式紧耦合于 Xen 虚拟机监控器，难以实现在其他虚拟机监控器平台上的移植，缺乏良好的通用性。DiskMig 内核模块原理如图 2 所示。

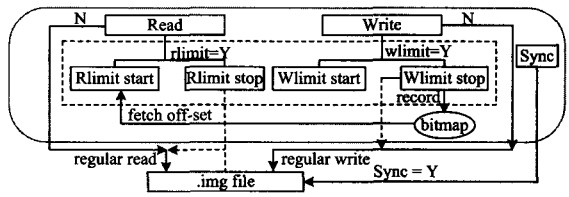


图 2 DiskMig 系统结构图

4.1.1 write-limit 模式

源端“预拷贝虚拟机磁盘”过程中，由于虚拟机对磁盘的 read 操作不会改变其状态，只需关心针对虚拟机磁盘的 write 操作，因此该阶段 DiskMig 系统进入 write-limit 模式，负责记录虚拟机对磁盘的 write 操作偏移位置(offset)，并将其存入 bitmap 结构。

4.1.2 read-limit 模式

“磁盘预拷贝”和内存迁移完成后，虚拟机在目的端恢复运行。此时 DiskMig 系统由 write-limit 模式切换到 read-limit 模式。a)对每个 read 型操作，负责检查其偏移是否存在于 bitmap 中。如是，进入读阻塞状态(rblock)，等待数据同步。如该位置处的数据已经到位，则直接进行读操作(read)。b)对磁盘某偏移(offset)处 write 操作，可直接对磁盘执行写操作。如该 offset 恰好位于 bitmap 结构中，且该 offset 处的信息尚未从源端传输至目的端，则通知源端放弃本次传输，以减少数据传输量。

4.1.3 Sync 模式

目的端收到源端发送的磁盘同步数据后，DiskMig 工作状态由 read-limit 模式切换到 Sync 模式，将相应磁盘差异同步到目的端磁盘，以保证虚拟机持续一致的运行状态。同时从 bitmap 中删除该位置信息。

4.2 bitmap-DiskMig 系统信息管理结构

DiskMig 系统 3 种不同工作状态下对磁盘 I/O 操作类型归结如表 1 所列。在虚拟机迁移过程中，需要处理的 I/O 信息量是巨大的。因此，合理地选择数据结构，决定了插入、删除及查找的效率，进而涉及整个迁移过程的效率。

表 1 DiskMig 3 种工作模式下的操作类型

write-limit 模式	read-limit 模式	Sync 模式
添加 Add	查询 Search	删除 Del

本文设计一种高效的存储管理结构 bitmap，来支持迁移过程中 I/O 信息管理，其原理如图 3 所示。

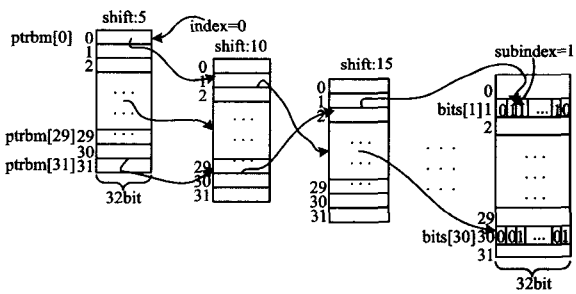


图 3 bitmap 原理结构图

设虚拟机磁盘共计有 N 个基本 block 块(4096kB)，bitmap 安排为由 32元素 int 数组和 bitmap 指针数组索引的联合构成的结构：将每个 block 块的偏移 pos 用 bitmap 中某

bits[index]的第 subindex 位表示(subindex=1 表示 write-limit 模式下某 pos 发生 write 操作,subindex=0 表示 pos 处无 write 操作)。如 N 或某个 pos 大小超出 bits[32]数组的表示范围(bits[32]可存储 32 * 32=1024 个偏移),为此引入多级索引,用 shift 表示索引级数,由上级索引指向下级 bits[subindex]变量。

定义图 4—图 6 的原子操作类型。

1) bit-size:=N;
2) bm_shift:=p(如果 2^p<bit-size<$2^{p+1}-1$,p=0,5,10,…)
3) is_muti-index:=((shift)! =5)
4) index:=(pos>>shift)
5) submask:=(1<<shift)−1
6) subindex:=(pos & submask)
7) set_bit(u,v):=(u|1<<(v))
8) has-bit(u,v):=(0! =(u&(1<<v)))

图 4 bitmap 原子操作图

```
Operation bitmap-set(bitmap,pos,val)
        /* val=1 表示 add 操作;
           val=0 表示 del 操作
           */
        shift:=bm_shift;
        index:=pos>>shift;
        subindex:=pos & submask;
        if(is_muti_index)
                bitmap-set(bitmap->ptrbm[index],sub-
        indx,val);
        else
                set_bit(bitmap->bits[index],subindex,
        val);
```

图 5 bitmap 操作 Add(Del)-operation

```
Operation bitmap-search(bitmap,pos)
        shift:=bm_shift;
        index:=pos>>shift;
        subindex:=pos &submask;
        if(is_muti_index && bitmap->ptrbm[index]! =
        NULL)
                bitmap-search(bitmap->ptrbm[index],sub-
        index);
        else
                return has_bit(bitma->bits[index],subin-
        dex);
```

图 6 bitmap 操作 Search-operation

基于以上基本操作,分析 bitmap 时间、空间复杂度。

时间复杂度分析:对于给定偏移 pos,只需由其 index 和 subindex 在 bitmap 中索引查找相应位置,然后将 bits[index]的相应 subindex 位置为 1(0)或判断 subindex 位是否为 1,因此时间复杂度为 O(1)。

空间复杂度分析:对于含 N 个 block($2p$<bit-size<$2p+1-1$,p=0,5,10,…)的虚拟机磁盘,用 bitmap 结构存储只需要 O(log$N$)空间。

表 2 bitmap 和不同存储类型性能比较

存储类型	时间复杂度 (add,del,search)	空间复杂度 (n 为 offset 个数)
有序数组	O(n),O(n),O(logn)	O(n)
无序数组	O(1),O(1),O(n)	O(n)
链 表	O(n),O(n),O(n)	O(n)
bitmap 结构	O(1),O(1),O(1)	O(logn)

4.3 磁盘差异同步算法

虚拟机迁移过程对虚拟机中应用程序运行的影响可归结为两个方面:虚拟机应用程序运行的透明性及运行总体耗时。针对前者,本文采取“在线迁移”方式,以尽可能减小因内存及磁盘迁移过程给应用程序带来的中断影响;针对后者,基于本文之前分析得出,虚拟机中应用程序的总体耗时主要取决于同步磁盘差异所需时间。本文设计了“Transfer on Demand”磁盘同步算法,根据迁移后虚拟机在目的端运行的动态需要同步磁盘差异。并设计了预测启发式算法——“Forward-ahead”,通过跟踪统计每个应用程序 I/O 操作的规律,预测程序执行过程中未来的 I/O 行为,快速同步磁盘差异,从而减少同步过程中反复的请求所带来的系统开销,缩短应用程序执行时间。

4.3.1 按需传输算法——Transfer on Demand

DiskMig 处于 read-limit 模式时,当目的端虚拟机由于磁盘差异而出现“读阻塞”时,如何快速同步磁盘差异,对虚拟机及其中应用运行效率有重要影响。本文提出 Transfer on Demand 及 Forward-ahead 算法,以实现快速同步。Transfer on Demand 算法描述如图 7 所示,其特点是:

1)在目的端同步磁盘差异过程中,并不禁止虚拟机对磁盘的 I/O 操作。

2)为避免传输不必要数据,不采取由源端将 write-limit 模式下所记录所有 write 内容发送给目的端的方法。

3)只有在目的端运行出现“读阻塞”时,才由目的端通知源端按需传输相应数据,并依据 Forward-ahead 方法进行传输优化。

```
/* After Vm migrate to destination */
        Source:transfer(bitmap);
        ransfer(blocks);
        Destination:keep_rum(Vm&&Vm_I/O);
While(true){
        if(need(block,pos,I/O_flag)){
                if(I/O_flag==READ){
                        ① Destination: demand(block,
                          pos);
                          Source:
                        ② { prior_trans(block,pos);
                          { forward(s,pos);
                          Destinatio;
                        ③sync(block + some_blocks_
                        follow);
                }
        }
}
```

图 7 Transfer on Demand 思想

4.3.2 预测启发式算法——Forward-ahead

在目的端 DiskMig 处于 Read-limit 模式时,虚拟机及其中应用对磁盘的 I/O 操作具有一定的规律,即 I/O 操作偏移具有阶段连续性。选用 Linux 下常见的编译任务“Util-linux

compilation",图 8 是其运行在 DiskMig read-limit 模式下 read 操作所对应的偏移 offset 的分布规律。

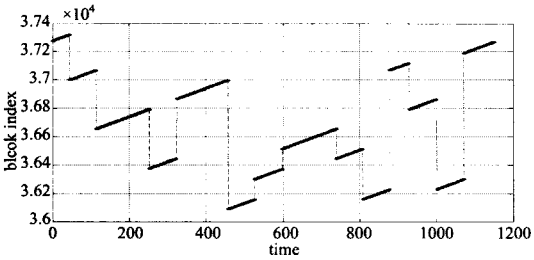


图 8　read-limit 模式 Compile Util-linux I/O 分布图

由图 8 可看出,在 read-limit 模式下,运行过程中由于 read 操作而产生的 rblock 状态将程序执行状态划分为若干相对分离阶段(图中由虚线隔开的 14 个部分),每个阶段 read 操作对应的 offset 近连续分布(图中斜线表示 offset 按序递增)。

因此,对应以上各阶段,可利用"TOD&FA"思想,源端负责优先传输相应部分的数据,并预测程序执行过程中下一阶段所需 I/O 数据,由本次最后一个 offset 起连续预传输若干磁盘偏移数据,以提高网络及 I/O 利用率,提高迁移效率。

4.4　DiskMig 系统实现原理

基于以上所述,对虚拟机磁盘迁移系统基本实现原理予以小结。本系统基于 Xen 虚拟机技术平台实现,在涉及迁移的源端和目的端节点安装并运行 Xen 虚拟机监控器(VMM),并安装 DiskMig 系统,使其运行在 Domain0 下,并与待迁移的虚拟机磁盘相关联。系统运行过程描述如图 9 所示。

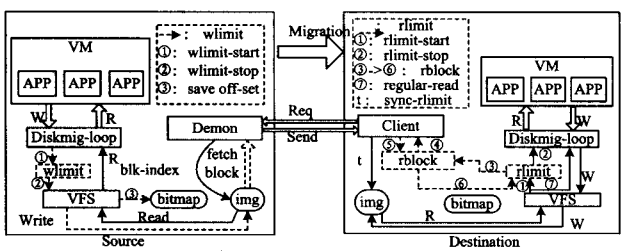


图 9　系统运行原理图

1)目的端计算资源准备就绪后,源端即开始迁移某台虚拟机实例。首先,利用 losetup 接口将 DiskMig 模块与虚拟机磁盘关联,以实现迁移过程中 DiskMig 对虚拟机磁盘的 I/O 控制。随后,开始虚拟机磁盘 pre-copy 过程,同时 source 端的 DiskMig 模块启动 write-limit 模式。对每个 I/O 操作,该模式判断其 I/O 类型。如果为 read 类型,执行正常的读操作;如为 write 类型,DiskMig 除完成 write 操作外,还将该 write 操作的偏移信息(off-set)保存到 bitmap 结构中(如图 9 左①－③所示)。

2)磁盘预拷贝和内存迁移完成后,首先将源端 write-limit 模式记录的 bitmap 信息传向目的端,随后虚拟机在目的端继续运行,DiskMig 模块切换到 read-limit 模式,同时由源端并行传输 write-limit 下的磁盘数据。对每个 write 操作,直接执行 write 命令(如图 9 右'W'部分所示);对 read 操作,如偏移位置 pos 不在 bitmap 结构中,执行正常的 read 操作(如图 9 右⑦所示);如其偏移位置 pos 位于 bitmap 记录中,且该 pos 处磁盘数据尚未由源端传输至目的端,则进入 read 操作,进入 rblocked 状态,同时根据所采用的"TOD&FA"方法由 client 端通知源端守护进程 demon,优先传输该 offset 位置处的数据(如图 9 右①－⑥所示);对从源端接收的 sync 数据,DiskMig 由 read-limit 转为 sync 模式,同步磁盘文件差异(如图 9 右 t 所示)。

5　实验结果及分析

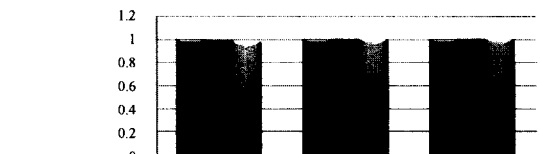
为了验证 DiskMig 系统及 Transfer on Demand 思想在迁移过程中的性能,设计两组实验,分别测量 DiskMig 系统在非迁移场景、迁移场景下的性能。

5.1　实验一:非迁移场景 DiskMig 迁移性能

本实验的目的是验证 DiskMig 系统只在一般模式(Normal)和 write-limit 模式下工作时对虚拟机中应用运行性能的影响。为充分测量 DiskMig 的 I/O 性能,本实验选取 3 类 I/O 密集型任务作为本实验测试负载,分别是 dbench 文件系统 benchmark、编译 iputils、编译 linux kernel。同时,为避免其他应用程序运行时影响磁盘 I/O,禁止待测虚拟机中其他应用的运行。本实验由如下两组对比实验构成。

1)测试采用 linux 系统原始 loop 驱动程序控制虚拟机磁盘 I/O 时,运行以上 3 类不同任务时任务所耗费的时间。

2)测试基于 DiskMig 系统控制虚拟机磁盘 I/O 时,运行 3 类不同任务时任务所耗费的时间。为了分析 DiskMig_loop 的不同模式对程序运行的影响,细化为两个 Test Cases:(1)测量 DiskMig 在 normal 模式下运行(如图 10 Migloop M1 所示);(2)测量 DiskMig 在 write_limit 模式下运行(如图 10 Migloop M2 所示)。



	dbench	compile iputills	compile kernel
Loop	1	1	1
MigLoop M1	0.97937568	0.999655933	0.99992165
MigLoop M2	0.979358286	0.999233992	0.9996332

图 10　非迁移场景 DiskMig 性能测试结果

由图 10 可得,极端测试条件下(dbench 测试实例)性能差异仅在 1.1%～2.1%以内,而真实应用(Linux kernel-compile)性能差异仅在 0.1%以内。这说明,DiskMig 只在一般 write-limit 迁移模式下解释磁盘 I/O 变化时,对虚拟机运行性能几乎不产生任何影响,基本可以忽略不计。

5.2　实验二:迁移场景 DiskMig 迁移性能

设计该实验的目的是测试磁盘迁移过程中,DiskMig 的 3 种迁移模式及所采取的磁盘差异同步算法对程序运行的影响。实验由两组对比实验构成,测试结果如图 11 和图 12 所示。

1)程序在不受任何限制下以原始状态运行;

2)在〈磁盘迁移,DiskMig 控制,磁盘差异同步算法〉三元素共存时程序的运行。分别测试在不预拷贝磁盘(DiskMig 对程序运行影响的极端情况)、预拷贝磁盘方式下 DiskMig 系统对虚拟机及其中程序运行的影响。选用编译 util-linux 和 tomcat 作为实验任务。首先在源端下载某编译任务的源代码〈write-limit〉,拷贝磁盘到目的端,然后在目的端对源码编译,此时 DiskMig 先后处于 read-limit 和 Sync 模式,并采用

Transfer on Demand 思想同步磁盘差异。本实验在网络带宽为 100M 的环境下进行。

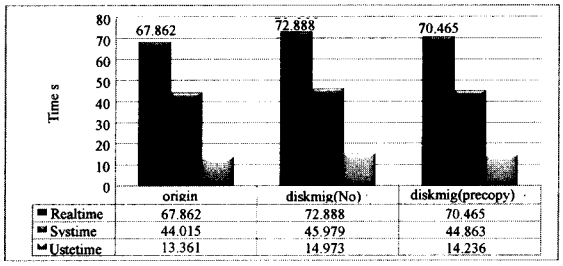


图 11 迁移场景下 DiskMig 性能测试结果 1

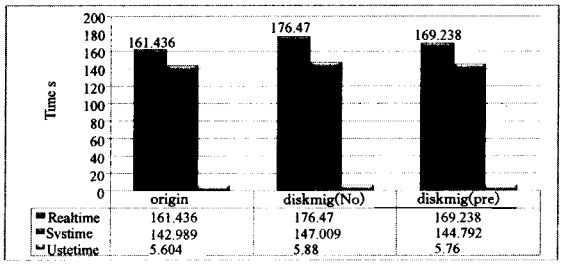


图 12 迁移场景 DiskMig 性能测试结果 2

综合以上两组实验可以得出：在迁移磁盘时，在〈磁盘迁移，DiskMig 控制，磁盘差异同步算法〉三元素作用下：

1)极端情况下的实验关于迁移时机的选择思路是：在源端完整下载完编译所需全部源代码之后，才将虚拟机磁盘从源端迁移到目的端开始编译。因此，编译过程依赖于源端磁盘下载的所有源码 I/O 数据，需通过网络进行磁盘差异同步。此时程序运行性能的影响 $\eta = \Delta T / T_{origin} = T_{disk(No)} - T_{origin} / T_{origin}$ 分别为 7.406%和 9.312%(10%以内)。又依以上分析，有 $\eta = \eta_1 + \eta_2$，其中 η_1 和 η_2 分别表示因网络传输和 DiskMig 系统及其同步算法带来的程序运行性能损耗。由表 3 得，在 100M 的网络条件下，两组编译用例所耗的 $\eta_1 = (\Delta s / 100M) / T_{origin}$（$\Delta s$ 表示需要同步数据量）分别为 2.210%和 1.858%。因此，仅由 DiskMig 系统及其同步算法带来的程序运行性能损耗 $\eta_2 = \eta - \eta_1$ 分别为 5.196%和 7.312%。

表 3 迁移场景 DiskMig 实验数据统计

	状态	性能损失 (%)	同步数据量 (MB)	同步速率 (个/s)
Compile	无预拷贝	7.406	10.4	436
util-linux	预拷贝磁盘	3.835	4.8	472
Compile	无预拷贝	9.312	22.0	373
tomcat	预拷贝磁盘	4.850	11.8	481

2)验证正常迁移场景下的实验采纳“单轮磁盘预拷贝”迁移方式。关于迁移时机的选择思路，是在源端磁盘下载编译所需源代码的过程中，即启动虚拟机磁盘迁移。这样虚拟机在目的端运行时，编译所需的部分源代码已经存在于目的端磁盘，仅需通过网络进行同步所需剩余部分磁盘差异。程序运行性能的影响 $\eta = \Delta T / T_{origin} = T_{disk(No)} - T_{origin} / T_{origin}$ 分别为 3.835%和 4.850%(5%以内)。同 1)$\eta = \eta_1 + \eta_2$，其中 η_1 和 η_2 分别表示因网络传输和 DiskMig 系统及其同步算法带来的程序运行性能损耗。由表 3 得，在 100M 的网络条件下，两组编译用例所耗的 $\eta_1 = (\Delta s'/100M) / T_{origin}$（$\Delta s'$ 表示需要同步数据量）分别为 0.735%和 0.743%，因此，仅由 DiskMig 系统及其同步算法带来的程序运行性能损耗 $\eta_2 = \eta - \eta_1$ 分别仅为 3.100%和 4.107%。

6 相关工作比较与结论

Xen Live Migration[4] 首次提出“在线迁移”思想，完成虚拟机内存状态迁移，取得较高迁移效率，停机时间只需几百毫秒。但该迁移方式不支持传输虚拟机磁盘，在迁移过程中源机和目标机通过共享网络磁盘(NAS 方式)[4] 方式访问磁盘，导致无法在非共享网络磁盘环境下实现迁移，应用范围有限。Robert Braford 等人[7] 提出了带虚拟机磁盘迁移的解决方案，其目标也是保持尽可能短的停机时间，但其不足在于虚拟机在目的端恢复运行时，须禁止一切 I/O 操作，须等到将源端磁盘写变化全部恢复到目的端后才能启动 I/O 操作，导致虚拟机中运行任务的 task-time 增加，I/O 利用率低。纵观以上两种迁移思路，仅限于追求尽可能短的停机时间，而忽略了虚拟机中应用程序在迁移过程中总的运行时间。

基于以上不足，本文提出了一种新的采用带虚拟机磁盘的迁移思路，设计并实现了具有 write-limit 及 read-limit 迁移模式的 DiskMig 系统。该系统通过 write-limit 模式解释虚拟机迁移过程中源端磁盘的 I/O 变化，并将其存储于具有良好时空复杂度特性的 bitmap 结构中；虚拟机迁移到目的端时，由 read-limit 模式负责检查磁盘的 I/O 操作是否位于 bitmap 的限制范围内，从而决定是否进行正常 read 操作或同步磁盘差异工作。尤其是在同步磁盘差异方面，本文在保证虚拟机可以进行正常 I/O 操作的前提下，采用“按需传输”及预测启发式算法(Forward-ahead)，快速同步磁盘差异，从而大大减小了对虚拟机中应用程序运行时间的影响。

实验证明，在网络带宽为 100M 的迁移环境中，通过 DiskMig 支持，采用 TOD&FA 算法来迁移虚拟机磁盘文件，可以取得很好的迁移性能(迁移过程对虚拟机及其中应用程序性能的影响仅为 5%左右)。

下一步工作考虑对不同应用的磁盘 I/O 特征建模，依据该模型增添同步磁盘差异时的预取功能，进一步优化同步磁盘差异的效率。此外，考虑将本系统扩展到广域网环境中，拓宽虚拟机迁移的应用场景。

参 考 文 献

[1] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[C]// Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles(SOSP19). ACM Press, 2003: 164-177

[2] Milojicic D, Douglis F, Paindaveine Y, et al. Process migration [J]. ACM Computing Surveys, 2000, 32(3): 241-299

[3] Nelson M, Lim B H, Hutchins G. Fast transparent migration for virtual machines[C]// Proc. of the USENIX Annual Technical Conf. 2005: 391-394

[4] Clark C, Fraser K, Hand S, et al. Live migration of virtual machines[C]// Proc. of the 2nd ACM/USENIX Symp. on Networked Systems Design and Implementation(NSDI). 2005: 273-286

[5] Travostino F, Daspit P, Gommans L, et al. Seamless Live Migration of Virtual Machines over the MAN/WAN. iGrid, 2006

(下转第 297 页)

4.3 人体模型关节位置的确定

人体黄金分割点只能大概表示关节点的位置，每个模型都有其自身的特点，人体的姿势也不一致，所以确定关节点的位置，才能提取出骨架。

Takuya Oda 等人提出一种基于测地距离的交互式骨架提取算法[6]，该算法不仅需要人工参与操作，而且计算复杂度比较大，但是提取的准确度比较高。采用该算法来确定人体模型关节点的位置，具体步骤如下：

1)用 Dijkstra 求出每个身体模块的测地距离。

2)计算出测地距离之和。

3)归一化测地距离，其计算式如下：

$$g_{v_{I\_normal}} = \frac{g_{v_i} - \min_{j\in n} g_{v_j}}{\max_{j\in n} g_{v_j}} \tag{4}$$

其中，$g_{v_{I\_normal}}$ 表示归一化的测地距离之和，g_{v_I} 表示每个点的测地距离。

4)根据归一化的测地距离和每个人体模块的关节点数目，把人体模块分成几个小组，两组之间的交界处确定出关节点。

由于有了关节点的大概位置，因此无需采用人工参与。我们又对模型进行精简和分割，仅对每个精简后的模块进行节点提取，大大减少计算复杂度。

将提取的关节点应用于原始模型，按图 2 所示的人体骨架模型进行连接，形成骨骼，最终形成骨架树。

5 实验结果

在 PC 机(操作系统为 Windows XP 专业版，CPU 为 Intel Core Duo CPU 1.60GHz，内存为 1GB)、C++和 OpenGL 图形库下进行实验并选择几个男性模型和女性模型作为实验数据。图 6(a)是比较典型的男性模型骨架提取的效果图，该模型原始的顶点数是 1504。图 6(b)是比较典型的女性模型骨架提取的效果图，该模型的原始顶点数为 28439。

(a)男性模型骨架提取效果图　(b)女性模型骨架提取效果图

图 6　人体模型骨架提取效果图

现有的骨架提取算法大多是针对所有模型进行提取，具有通用性，而我们的算法是专门对人体模型进行骨架提取的。针对三维人体模进行骨架提取，在提取效果和计算时间上占一定的优势。与文献[7]的 Ilya 算法计算时间之对比如表 1 所列。

表 1　算法的计算时间

模型顶点数	Ilya 算法计算时间	本文算法计算时间
19001	12.6 秒	0.5 秒
34339	56.8 秒	1.1 秒
56856	77.1 秒	1.4 秒

结束语　本文提出一种三维人体模型的骨架提取算法。本算法首先对模型进行精简，然后采用人体的特点和黄金分割律获得大概的节点位置，采用测地距离算法对模型的节点进行修正，确定其位置。针对人体模型进行骨架提取，与现有的算法相比，大大节省提取的时间。该算法目前只针对人体模型进行骨架提取，下一步将继续努力，使它能应用于动物模型的骨架提取。

参 考 文 献

[1] 庄越挺，刘小明，潘云鹤. 一种基于视频的人体动画骨架提取技术[J]. 计算机研究与发展，1999，37(4)：498-506

[2] Adams J. DirectX 高级动画制作[M]. 刘刚，译. 重庆：重庆大学出版社，2005

[3] Teichmann M, Teller S. Assisted Articulation of Closed Polygonal Models[C]// Proc. 9th Eurographics Workshop on Animation and Simulation. 1998：254-268

[4] Verroust A , Lazarus F . Extracting Skeletal Curves from 3D Scattered Data[J]. The Visual Computer，2000，16(1)：15-25

[5] Hisada M, et al. A 3D Voronoi-based Skeleton and Associated Surface Feature[C]// Pacific Conference on Computer Graphics and Applications. 2001：89-96

[6] Oda T , Itoh Y , Nakai W , et al . Interactive Skeleton Extraction using Geodesic Distance[C]// Artificial Reality and Telexi stence—Workshops，2006. ICAT '06. 16th International Conference. Nov. 2006：275-281

[7] Baran I, Popovi′c J. Automatic rigging and animation of 3d characters[J]. ACM Transactions on Graphics (SIGGRAPH 2007)，2007，26(3)

[8] 张雄. 黄金分割的美学意义及其应用[J]. 自然辩证法研究，1999，15(11)：5-8

[9] 黄春峰. 人体科学与"黄金分割律"[J]. 发明与革新，2000，7：36

[10] 张嘉. 人体美的黄金比例[J]. 医药与保健，2006，4：50-51

[11] Hilaga M, Shinagawa Y, Komura T, et al. Topology matching for fully automatic similarity estimation of 3D shapes [A]// Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH[C]. Los Angeles, California，2001：203-212

[12] Chen Z, Lee H J. Know ledge-guided visual perception of 3-D human gait from a single image sequence[J]. IEEE Trans. on Systems, Man, and Cybernetics，1992，22(2)：336-342

[13] Hoppe H. Progressive mesh[C]// Proc. of ACM SIGGRAPH, Computer Graphics Annual Conference Series. 1996：99-108

(上接第 261 页)

[6] Kozuch M, Satyanarayanan M. Internet suspend/ resume[C]// Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications. 2002

[7] Bradford R , Kotsovinos E , Feldmann A , et al . LiveWide-Area Migration of Virtual Machines Including Local Persistent State [C]// ACM/Usenix International Conference on Virtual Execution Environments(VEE'07). June 2007

[8] Warfield A , Hand S , Fraser K , et al . Facilitating the Development of Soft Devices[C]// USENIX. 2005

[9] http://linuxcommand.org/man_pages/losetup8.html

[10] Osman S, Subhraveti D, Su G, et al. The design and implementation of zap: A system for migrating computing environments[C]// Proc. 5th USENIX Symposium on Operating Systems Design and Implementation(OSDI-02). 2002：361-376

[11] Sapuntzakis C P, Chandra R, Pfaff B, et al. Optimizing the migration of virtual computers[C]// Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI-02). December 2002