doi:10.3772/j.issn.1002-0470.2016.03.001

基于等差划分的虚拟机实时迁移技术①

邹庆欣②*,**,*** 郝志宇③*** 云晓春*,*** 王冲华**,***

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

(***中国科学院信息工程研究所 北京 100193)

摘 要 研究了虚拟机的实时迁移。对局域网内存迁移的预拷贝算法、后拷贝算法以及由这两种算法构成的混合拷贝算法的特性进行分析,在改进混合拷贝算法的基础上提出了基于等差划分的虚拟机实时迁移技术,该技术把第一轮传输的内存页划分成近似等差数列形式的多段,以减少第二轮传输的内存页。不同工作负载条件下的试验表明,与混合内存拷贝方式相比,基于等差划分的虚拟机实时迁移技术平均减少了25%的同步内存位图时间,29%的后拷贝方式发送的内存页以及2.2%的总迁移时间,在一定程度上提高了混合内存拷贝方式的虚拟机实时迁移性能。

关键词 实时迁移,虚拟机,预拷贝,后拷贝,混合拷贝,等差划分

0 引 言

云计算大数据时代到来的同时,牢牢地捆绑了虚拟化技术。虚拟机更是虚拟化技术的一个最直接而且集中的体现。虚拟机依靠软件模拟的方式拥有物理计算机的全部硬件特征。而且多个虚拟机还可以同时在一台物理机上彼此互相隔离地运行。不仅如此,虚拟机在运行时,在一定条件下还可以在不同的物理主机之间来回移动,用以增加虚拟化的灵活性。这被称为虚拟机实时迁移技术。实时迁移技术在虚拟机众多技术中无疑是一项卓越而实用的服务。虚拟机实时迁移有广域网迁移和局域网迁移之分。在广域网迁移中不仅要迁移虚拟机内存,还要迁移虚拟机的磁盘镜像文件及网络状态等。而局域网迁移,主要针对内存,不必迁移虚拟机磁盘镜像文件等,完全可以利用 NFS 等服务采用共享的方式。本文主要研究局域网迁移中的内存迁移。

在虚拟机局域网内存迁移技术中,预拷贝算法是最常用,也是最实用的算法之一。但是预拷贝算法对于读操作密集型和写操作密集型的虚拟机应用负载来说,效果是截然不同的,对于预拷贝算法来说,在写操作密集型的应用负载运行时,其往往达到最大迭代轮数而失效。后拷贝算法与预拷贝算法正好相反,并且只传输一轮内存页,而且对于写操作密集型的负载很有效,但对于读操作密集型的负载却容易延长迁移时间。为了弥补预拷贝算法的缺点,一种结合了后拷贝算法的混合内存拷贝算法应运而生。这种算法只进行两轮内存页的传输,第一轮预拷贝式地推送全部内存页,而第二轮后拷贝式地拉求部分变脏的内存页,充分发挥了预拷贝和后拷贝算法的优点。本文对这种混合内存拷贝方式的迁移机制进行了研究,并提出了进一步的改进措施。

① 国家自然科学基金(61003261)和国家科技支撑计划(2012BAH46B02)资助项目。

② 男,1982 年生,博士生;研究方向:虚拟化技术;E-mail: zouqingxin@nelmail.iie.ac.cn

③ 通讯作者,E-mail: haozhiyu@iie.ac.cn

(收稿日期:2015-10-20)

1 相关研究工作

文献[1]首次提出实现了虚拟机实时迁移机制,并采用了文献[2]中所提出的预拷贝算法。预拷贝算法是虚拟机实时迁移中最常用的算法。主流的虚拟化平台均采用预拷贝算法。预拷贝算法的迭代过程如图 1 所示[9]。

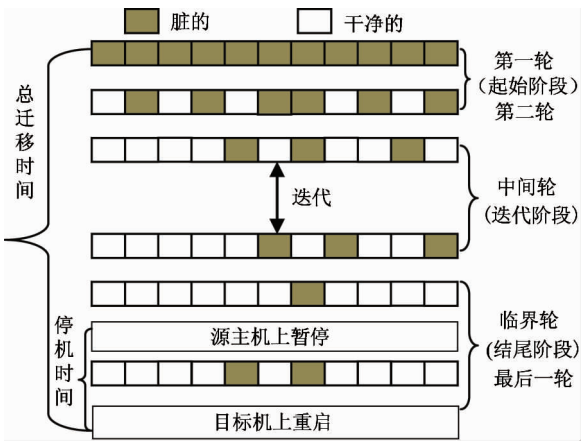

图 1 预拷贝算法迭代过程

预拷贝算法试图最小化最后一轮中传输的内存页,以保证停机时间尽可能短。在这个过程中位于源主机上的虚拟机物理内存镜像,通过网络被传输到目的主机,同时源主机持续运行。所以为了保持虚拟机内存状态在源主机和目标主机之间一致,一些发送到目标主机的内存页,在重新访问修改变脏之后应该被再次发送到目标主机。因此预拷贝算法用一个迭代的机制来不时地检查变脏的内存页并重传它们。

在虚拟机迁移的各种算法中,针对预拷贝算法的研究相对较多,文献[3-6]分别介绍了不同的压缩机制和方法对预拷贝算法的改进。同时文献[7-9]从内存页不同传输顺序的角度改进了预拷贝算法。文献[10-12]则从调节虚拟机 CPU 的角度改进了预拷贝算法。

除了预拷贝算法以外,文献[13]提出了后拷贝算法,与预拷贝方式相反,后拷贝技术首先从源主机端发送 CPU 状态和虚拟机能够恢复运行的最小工作集到目标主机,然后虚拟机在目标主机端被重启。随后,虚拟机在目标主机端启动运行过程中,其所需的但并不存在于目标主机的内存页从源主机端被请求过来或被推送过来。与预拷贝方式相比,后拷贝算法减少了总迁移时间,但增加了停机时间。

借鉴于各自的优缺点,文献[14]实现了内存混合复制方式的虚拟机实时迁移机制。其依次执行全内存同步、内存位图同步和脏内存同步三个过程,与预拷贝算法相比,该机制同时降低了停机时间和总迁移时间。

专门针对混合算法的改进工作也有相关的研究。文献[15]增加了预学习阶段,估计出变脏比较频繁的内存页,并辨认出工作集,从而在第一轮省去工作集的传输,而只在第二轮后拷贝阶段传输,从而达到减少内存页传输的目的,而且研究了后拷贝阶段的安全性和可靠性。

文献[16]的研究中,在停机阶段发送缓存位图和脏页位图两种位图,对于在两个位图中的页面,采用压缩后变量传输的方式,从而改进了混合方式的内存传输。

文献[17]提出了一种更接近于后拷贝方式近似于混合方式的迁移技术,在停机以前其并不传输任何内存页,只是在停机时间内,传输了更大的工作集,不光是写频繁的,就连读和接触的内存页也一并传输过去。

还有一些其它方式的迁移技术,例如文献[18]基于日志采用检查点/恢复和跟踪/重放的技术,该技术类似于预拷贝算法,但传输的不是内存而是日志,因此数据传输量较小,但实现相对比较复杂。文献[19]采用了两个线程并行传输,第一个线程从头至尾传输所有内存页,同时另一个线程传输变脏的内存页,当第一个线程遵循时间限制结束时,立刻进入停机阶段。

总之,现有的研究工作大部分针对预拷贝算法,另外也提出了一些其它的方法。针对混合方式迁移技术的研究还不是很多,有待研究人员进一步地改进。

2 混合内存复制方式

主流虚拟化平台上均采用预拷贝算法作为虚拟

机实时迁移的默认算法,然而当虚拟机上负载较高时,预拷贝算法往往达到最大轮数而失效。

采用混合内存复制方式的虚拟机实时迁移技术通常不会遇到失效问题。混合内存复制方式的虚拟机实时迁移技术的特点就是不像预拷贝算法那样迭代多次,而只是重传一次,只进行两轮传输,并且第二轮采用后拷贝的形式,混合内存复制方式将整个内存同步过程分为全内存同步阶段、内存位图同步阶段和脏内存同步阶段三个过程。图 2 描述了混合内存复制算法的执行过程。从图中可以看到,混合内存复制方式只传输了一轮就暂停了虚拟机,而且在虚拟机暂停之前要把全内存同步到目的主机;第二阶段则要把记录内存变脏与否的位图同步到目的主机;第三阶段目标端向源端请求和内存位图相对应的变脏的内存页。混合内存复制方式,虚拟机暂停后传送的只是位图,避免了内存页的传输,从而避免了高负载情况下预拷贝算法失效的情形。但是由于第三阶段采用了后拷贝拉的方式,从而又增加了安全性的问题,因为后拷贝方式虚拟机在目标主机上已经重启,一旦虚拟机迁移失败,源端虚拟机将不可恢复,所以减少了第三阶段传输的内存页数目和加速第三阶段的过程,这是混合内存拷贝方式的一个有待改进的地方。本文着重研究了这个问题,并提出了行之有效的改进方法。

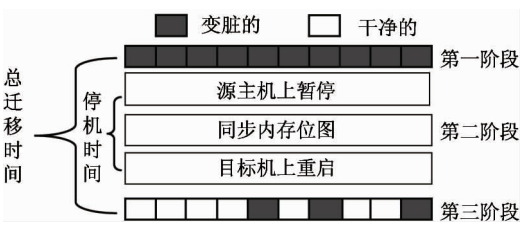

图 2　混合内存复制方式

3　等差划分算法设计

3.1　划分成不同的段

对于混合内存拷贝方式来说,只重传一次,且第一轮传输全部内存页,而且第二轮采用后拷贝方式传输的内存页是相对第一轮起始点变脏的内存页,包括相对第一轮传输点未变脏的内存页。这就会产生一种现象,即一个内存页在起始点和传输点之间被更改时,尽管相对于起始点来说是变脏的,但是相对于传输点来说是没有变脏的,对于这样的内存页,混合内存拷贝方式在后拷贝阶段是要传输的。但实际上这样的内存页是不需要传输的,因为该内存页在传输点以后没有被更改,换句话说只有在传输点和结束点之间发生变化的内存页才真正需要传输(图 3)。

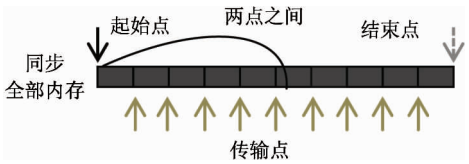

图 3　起始点,结束点和传输点位置

实际上,混合内存复制方式重传的一次忽略了这一细节。经过上述分析,可以说在第二轮后拷贝阶段是存在三种内存页的,如图 4 所示。

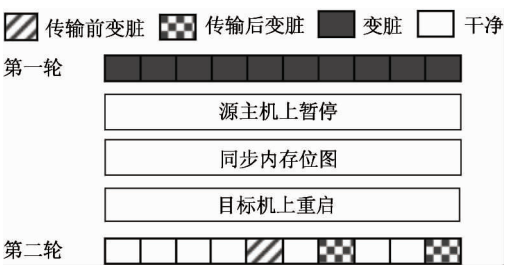

图 4　第二轮三种内存页面

在原来的混合内存拷贝方式中,第二轮只区分变脏的和干净的两种内存页,没有对变脏的内存页做具体的划分,而实际上变脏的内存页是可以区分成传输前变脏和传输后变脏两种情况的。完全可以通过有效的方法来尽可能多的找出这些相对传输点来说没有变脏的内存页,从而减少第二轮后拷贝阶段传输的内存页(图 5)。

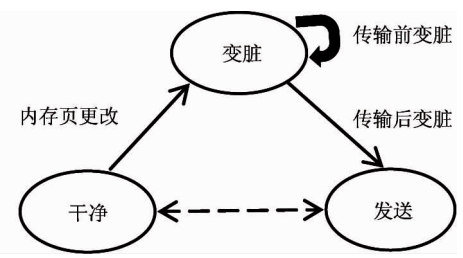

图 5　第二轮三种页面关系

实际上，以 4kB 大小的内存页为例，一个 512MB 内存的虚拟机就有 131072 个内存页，统计每一个内存页在传输前或后的状态是不现实的，将拖延迁移的时间。一个有效的方法就是改点为段，把内存进行划分，统计一段内存传输前或后的变化情况（图 6）。

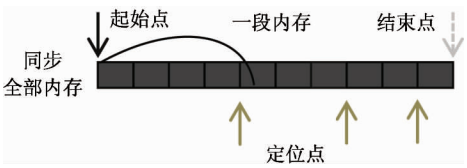

图 6　起始点、结束点和定位点位置

对于混合内存传输方式来说，只有两轮，第一轮传输全部内存页，同时第二轮传输少量内存页，第一轮的分段显的很关键。段选择的过大，很容易遗漏一些传输前变脏而传输后没有变脏的内存页，发送一些没有必要发送的内存页，段选择的太小，又会使段的数量增多，从而增加计算时间和迁移的时间。

划分内存成段的点称之为定位点，在每个定位点可以截取一次全部内存的变脏情况。在第一个定位点位置可以截取从起始点到当前定位点这段时间内全内存变脏的情况。以后每个定位点位置可以截取前一个定位点到当前定位点这段时间内全内存的变脏情况。最后通过把所有定位点截取的全内存变脏情况进行综合统计，来确定每个内存页到底是其所在的段传输前变脏还是传输后变脏。定位点的多少及其位置的不同引起了各个段之间的微妙关系，定位点可以是随意的无序的。但是从有序的角度讲，定位点引起的段划分一般可以分为倍分、差分和均分三种情况。（见图 7）

图 7 中左面三个点条线描绘了比例为 2，同时最小内存段分别为 3 单位、2 单位和 1 单位的倍分曲线变化关系。相同段数的情况下，最小内存段越大，则能表示的总内存越大，随着段数的增多，内存段也越来越大，相邻两段内存的差距也逐步增大。

同时图 7 右面三条虚线也描述了每段内存长分别为 7 单位、4 单位和 1 单位的均分情况。相同段数的情况下，每段内存越长则能表示的总内存越大。

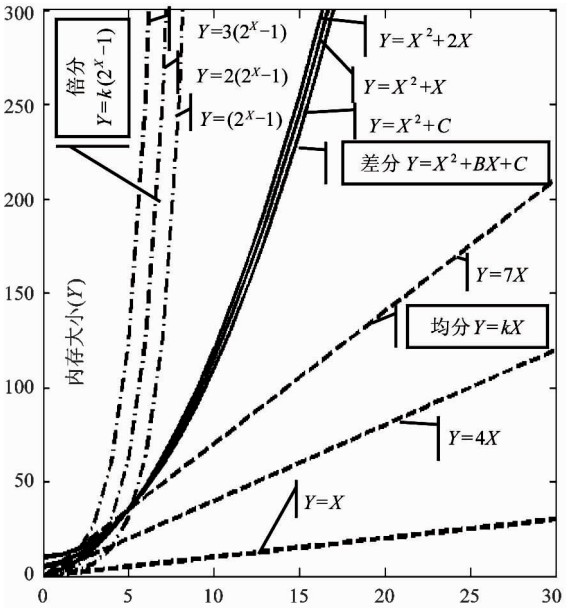

图 7　段长选择

但是相同段数的情况下，倍分最小段内存长和均分每段内存长相等时，倍分的总内存远远大于均分的。随着每段内存长的增大，均分线越来越陡峭，要赶上差分线和倍分线的变化趋势时，每段内存长要很大，而且段长比较单一，体现不出渐进式的变化，对变脏率比较大的小段内存非常不利，不容易辨认出小段内存中内存页具体是发送前变脏还是发送后变脏的。

最后图 7 的中间 3 条实线也绘制了最小内存段长分别为 1 单位、2 单位和 3 单位，公差为 2 单位的 3 条差分线。而且这 3 条差分线靠的比较紧凑，变化趋势基本一致。差分相对于倍分还是比较稳定的。总体上来说差分的最小段和倍分的最小段以及均分的每个段相等时，差分的内存变化率处于中间的位置。

总之，大段均分容易冗余传输前变脏的内存页，小段均分又会增加计算时间拖延迁移。倍分时，随着段数的增加相邻两段内存的差距越来越大，变化的不平稳。而差分的方式不仅可以平稳较快地收敛，同时又能区分小段内存的变化情况及不拖延迁移时间，是理想的段划分策略。

接下来介绍差分的表示公式，设等差数列 a_1，a_2，…，a_n 前 n 项的和为 S_n，公差为 d。则 $S_n = An^2 + Bn$，其中 $A = d/2$，而 $B = a_1 - d/2$。

在实际的运算中应当使 a_n 是大于零的整数,同时使 n 的取值尽可能大,所以选取 $d=2$,$A=1$,同时使 n 等于 S_n 的平方根后取整。并使 $St=n^2$,这样的话,分成两种情况:一是 $St=S_n$,在这种情况下 $B=0$,a_1 等于1;二是 St 不等于 S_n,在这种情况下完全可以增加临时变量 C,同时求整使 $B=(S_n-St)/n$,并求余使临时变量 $C=(S_n-St)\%n$。且 $0<(S_n-St)<(2n+1)$,因为当 $(S_n-St)=(2n+1)$ 时,$S_n=(n+1)^2$ 这与已知 n 等于 S_n 的平方根后取整是相互矛盾的,n 不可能等于 $n+1$。所以根据 (S_n-St) 的取值范围,B 只能取值0或1或2,而且 C 小于整数 n。当 $B=2$ 时,C 只能取0。当 $B=0$ 时,C 不能为0,如果为0实际上就是上述第一种情况了。总之当 C 不等于0时,说明在等差数列的基础上增加一个元素 C 而已。再把临时变量 C 按其值的大小顺序插入等差数列 $a_1,a_2,\cdots,a_n$ 中。以上就是等差划分的计算公式。内存经过这样的等差划分成段以后,内存变脏的情况就可以在每个定位点被截取。

3.2 等差划分整体流程

对于第一轮每个位置上的内存页来说,存在着变脏率和传输后的延迟时间两个要素,而变脏率越大则其再次变脏的几率也越大,同时传输后的延迟时间越长则其再次变脏的几率也越大。而变脏率和变脏的次数又是紧密相连的[9],如果把一个内存页变脏的次数看成高度,延迟的时间看成长度,就能得到一个几何抽象。

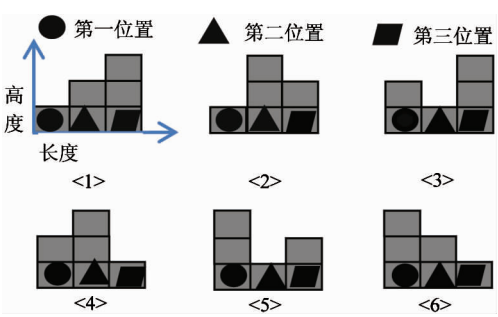

图8 变脏次数与延迟时间

从图8的几何抽象中可以看出:第一位置传输的内存页的延迟时间最长为3单位,第二位置为2单位,第三位置为1单位。同时把变脏次数不同的三个内存页分别放在不同的位置,总共有六种放法:

对<1>:长度 A(3, 2, 1)·高度 B(1 2 3) = 10。

对<2>:长度 A(3, 2,1)·高度 B(1 3 2) = 11。

对<3>:长度 A(3, 2, 1)·高度 B(2 1 3) = 11。

对<4>:长度 A(3, 2, 1)·高度 B(2 3 1) = 13。

对<5>:长度 A(3, 2, 1)·高度 B(3 1 2) = 13。

对<6>:长度 A(3, 2, 1)·高度 B(3 2 1) = 14。

从所做的数量积计算结果来看:应该避免延迟时间和变脏次数两个要素当中大的数值相遇,从而减少内存页传输后再次变脏的机会。

采用增加预处理阶段的方式,并统计预处理阶段内存页变脏的次数,使第一轮传输的内存页按照预处理阶段统计的变脏次数由低到高的顺序传输,就可以减少后拷贝阶段变脏的内存页数目。

具体地,统计变脏次数也是配合了在第一轮所进行的等差划分。在预处理阶段,根据全部内存的等差划分情况,每隔一段时间记录一次全部内存页变脏的情况,间隔的时间也是等差划分的。相同间隔的情况下,每次记录的变脏的内存页个数差不多,而等差间隔的情况下,每次记录的变脏的内存页个数一般不同,从而有利于根据变脏次数的高低而按序传输。时间复杂度为 $O(n)$ 的计数排序的方法用于排序内存页以节省时间。预处理阶段基本时间是固定的,然后以这个基本时间和每个内存段长的乘积作为一次间隔时间。

经过等差划分以后,每个定位点记录了内存页在每段传输时间内的变脏情况,所以在虚拟机暂停以前可以同步一次内存位图,这样就形成了事实上分两次同步内存位图的情形。第一次同步内存位图在虚拟机暂停之前,是从起始点到最后一个定位点这段时间相对于传输点来说变脏情况的内存位图。而第二次同步内存位图是发生在虚拟机暂停后,是从最后一个定位点到虚拟机暂停后这段时间内存变脏情况的位图。这样第二次同步内存位图时,由于

变脏位的减少,目标端设置缺页的行为减少,从而可以减少虚拟机暂停后同步内存位图的时间。

所以基于等差划分的虚拟机实时迁移机制实际形成了图 9 所示的流程,接下来将给出具体算法。

图 9 整体流程

3.3 等差划分整体算法

经过上面的分析讨论之后,具体的算法由表 1 给出。

表 1 基于等差划分的整体算法

```
定义基本时间变量 utime。
定义总内存页个数 p2m_size。
定义等差划分数组 G[ ]。
定义一次性传输的最大内存页数目 BATCH_SIZE。
初始化变量 TTlevel 为 0;
初始化变量 tv 为 BATCH_SIZE;
初始化变量 gg 为 0;
初始化变量 utime 为 100;
初始化变量 tlevel 为(p2m_size/BATCH_SIZE);
初始化变量 root1 为 int(sqrt(tlevel));
初始化变量 root2 为 root1 · root1;
if(变量 root2 和 tlevel 相等)
  {
    开辟空间使数组 G 的大小为 root1;
    使数组 G 的最后一个元素为 1;
    从后至前对数组 G 赋值使前一个元素比后一个大 2;
    赋值 level 为 root1;
  }
else
  {
    赋值 root3 为(tlevel 减去 root2);
    赋值 root4 为(root3 除 root1 所得整数);
    赋值 root5 为(root3 除 root1 所得余数);
    开辟空间使数组 G 的大小为(root1 +1);
    赋值倒数第二个元素 G[root1 -1]为 root4 +1;
    赋值最后一个元素 G[root1]为 root5;
    从倒数第三个元素 G[root1 -2]开始,使前一个元素比后面的多 2;
    移动最后一个元素 G[root1],使整个数组按从大到小的顺序排列;
    赋值 level 为(root1 +1);
  }
for(p =0;p <level;p + +)
    {赋值 btime 为(G[p] ×utime);
    延迟 btime 微妙时间;
    记录内存变脏位图至 all[p],并清零变脏位图;
}
  通过 all[ ]统计每个内存页变脏的次数并记录到 A[ ]中;
  根据 A[ ]中每个内存页的变脏次数由低到高的顺序重新排列内存页,把原位置记录到 D[ ]中;
for (n = 0; n < p2m_size;n + + )
  {
    赋值 E[n]为 gg;
    if(n 等于(tv1 · tv))
  {记录内存变脏位图到 seg[gg],并清零变脏位图;
    使 TTlevel 增加 1;
    使 tv1 增加 G[TTlevel];
    使 gg 增加 1;
  }
  发送 D[n]位置内存页;
  }
for ( m = 0; m < p2m_size; m + + )
  {
  for(z =E[m];z <gg;z + + )
  {
    if(seg[z]的 D[m]位为 1)
      {使 s_send 的 D[m]位为 1,并跳出本次循环;}
    }
  }
同步 s_send 位图到目标端;
暂停虚拟机;
用 SHADOW_OP_CLEAN 操作至 l_send;
同步 l_send 位图到目标端;
合并 l_send 与 s_send;
后拷贝方式发送内存页;
```

具体实现过程中,为了真实地展示等差划分算法的实际效力,在混合内存复制的第一轮,任何的优化措施都没有被采用,即使是全零页面,也是原原本本地发送过去,没有进行压缩。只是在第二轮采用了预取页和类似定时推送的优化机制。同时算法把预处理阶段的基本时间 utime 取值为 100μs。太大

的话,会增加总迁移时间,太小又不能产生效果。

4 试验及分析

为了验证等差划分算法的有效性,在 64 位操作系统 Centos-6.4 和 Linux3.0.0 内核以及 Xen-4.1.2 虚拟化平台上实现了该算法,并与混合内存拷贝方式进行了比较。每一个试验结果取 3 次试验的均值。

4.1 试验环境

目标主机和源主机是同一种机型,均为联想启天 M4300,其 CPU 类型都为 Intel(R) Core(TM) i3-2120 @ 3.30GHz,内存大小 4GB。两台主机由一台百兆 24 口的 TP-LINK 交换机连接。同时利用 NFS 服务把存在于目标主机上的虚拟机磁盘映像共享给源主机。客户虚拟机运行 Ubuntu-9.04 操作系统,同时保持每个物理主机最多只有一个虚拟机运行,并更改了 Xen-4.1.2 虚拟化平台上内存迁移的批次大小,由 256 个内存页取代原来的 1024 设置。

4.2 工作负载

试验中为了验证等差划分算法的有效性,选用了五种不同类型的工作负载:

(1)Idle:没有特别的应用程序在其上面运行,这个场景被用做比较参考,代表空闲类型应用。

(2)Nbench:这是一款用来对 CPU, FPU 和内存系统进行性能测试的工具。其本身是单进程的,试验中 shell 脚本被编写为用来同时循环运行 50 个 Nbench 的基准测试程序,同时为了避免长时间集中于某一项测试,控制某一项测试最少运行时间的 MINIMUM_SECONDS 参数由 5 秒被调减为 1 秒。

(3)Sysbench:该性能测试工具可以执行 CPU、内存、数据库等方面的性能测试。试验中配置参数—test = memory 来执行内存测试,线程数为 10 个,每个块大小为 256MB,总传输数据量为 5GB。

(4)Webbench:这是一款网站压力测试软件,它能够模拟 http 请求。目标主机配置 Tomcat 服务器和一个 50KB 的静态网页,同时虚拟机并发运行 10 个 Webbench 客户请求该网页。

(5)Dbench:该软件能够产生输入和输出负载给一个文件系统施加压力,试验中 5 个客户进程被配置用来产生负载。

测试中虚拟机被分配了 512MB 内存,并配置了一个 VCPU,同时虚拟机在源端运行时的相关工作负载参数也被测试出来,具体如表 2 所示。

表 2 各负载参数

负载	CPU(%)	网速	磁盘	内存(%)
Nbench	95 ~ 100	0	0	0 ~ 5
Sysbench	0 ~ 25	0	0	50 ~ 55
Webbench	75 ~ 85	60 ~ 80MB	0	0 ~ 1
Dbench	5 ~ 10	0	5 ~ 20MB	0 ~ 5

对于 Idle 场景来说,没有必要测量,而其它场景都是在 Idle 场景的基础上运行了特别的应用。对于 CPU 和内存的测试,利用了虚拟机操作系统自带的 top 工具,而对于网络速率和磁盘的性能则采用了 iftop 和 iotop 工具进行观测。从 CPU 角度来看,大部分情况下,Nbench 负载单进程时就可以占用 95% 至 100%,而其它工作负载单进程时都没能达到 90%,所以 Nbench 可以代表 CPU 密集型应用;从网络流量的角度来看,只有 Webbench 这一工作负载有网络流量,而且网速主要集中在每秒 60MB 至 80MB 之间,其余场景基本为 0,所以 Webbench 可以代表通信密集型应用;从磁盘角度来看,只有 Dbench 场景对磁盘有读写操作,而且大部分时间达到了每秒 5MB 至 20MB 的速率,其余工作负载几乎为 0,所以 Dbench 可以代表磁盘密集型应用;从内存的角度来看,Sysbech 负载的内存利用率最高,大部分情况下达到了 50% 到 55%,而其它工作负载的利用率远低于 Sysbench,所以 Sysbench 负载可以代表内存密集型应用。

4.3 工作负载测试

在这一部分,介绍了各种工作负载的测试,并比较了混合内存拷贝方式和本文提出的基于等差划分的虚拟机实时迁移技术两者的总迁移时间等参数,顺带也测试了 3.1 小节图 7 中所描述的系数 k 为 1 的倍分(TM)和系数 k 为 2 的均分(EQ)的情况用作比较,混合内存拷贝方式可以用 Hybrid 来指代,而等差划分技术可以用 DED 来指代。

图 10 给出了各种算法在不同工作负载情况下后拷贝方式传输的内存页数量情况。从图中的数据可以看到:与混合内存拷贝方式相比,等差划分的方法使 Idle 场景减少 36%,Nbench 场景减少 36%,Dbench 场景减少10.7%,Webbench 场景减少57.8%,Sysbench 场景减少2.4%,平均而言减少 29%。同时由于均分方式增加了第一轮的时延,很多场景下出现了后拷贝传输的内存页高于混合拷贝方式的情形,倍分方式由于测量数据的不稳定性等因素也出现了和均分方式相同的状况,但出现的几率明显小于均分的,而差分方式则没有出现这种状况。还要说明的是等差划分的方法只是减少了第二轮传输的内存页数,而几种算法第一轮预拷贝方式传输的内存页数目是相等的,都是把全部内存发送了一遍。

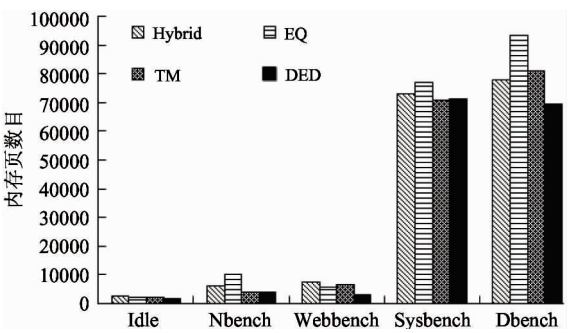

图 10　后拷贝方式传输内存页数

在 xen-4.1.2 虚拟化平台上,虚拟机在源主机上执行暂停操作时,其暂停时间不固定,有一定的变化范围,而且暂停时间远大于同步内存位图的时间,所以迁移时的停机时间不能准确反映出同步内存位图的时间变化,鉴于此,在具体的试验中,只是截取了虚拟机在源主机上暂停后到虚拟机在目标主机上重启前同步内存位图所需要的时间。从图 11 中可以看出,对于不同负载来说,同步内存位图的时间和后拷贝方式内存页的数目变化趋势不一样,并不是后拷贝内存页数目越多,同步内存位图的时间就越长,而是随着负载类型的不同而有所不同;但是对于同一负载来说,后拷贝方式内存页数目的下降,会带来同步内存位图时间的下降。与混合内存拷贝方式相比,等差划分的方法使 Idle 场景减少 23.1%,Nbench 场景减少 31%,Dbench 场景减少 28%,Webbench 场景减少 23%,Sysbench 场景减少 22%,平均而言减少 25%。而且倍分和差分的情景,也都出现了下降的趋势,不受后拷贝方式发送的内存页数量的影响,可见分两次同步内存位图的方法在减少虚拟机暂停后同步内存位图的时间的这个问题上起到了关键性的作用。

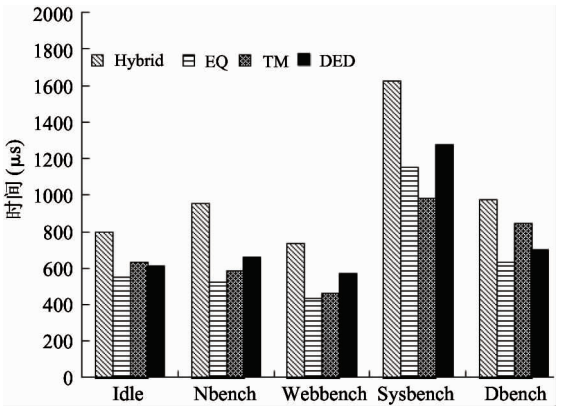

图 11　同步内存位图时间

图 12 对比了总迁移时间。等差划分的方法使 Idle 场景减少 0.3%,Nbench 场景减少 1%,Dbench 场景减少 4%,Webbench 场景减少 5%,Sysbench 场景减少 0.7%,平均而言减少 2.2%。尽管等差划分的方法使第二轮传输的内存页数目有所减少,但第二轮发送的内存页数目只占全部内存很少部分,而且又增加了预处理阶段,并使算法延迟。所以总迁移时间计算下来,改进的比例不是很高。而均分的方法,由于第一轮分的段数过多,明显增加了总迁移时间,而倍分的方法基本和混合方式持平。可见采用差分方式还是有一定积极作用的。

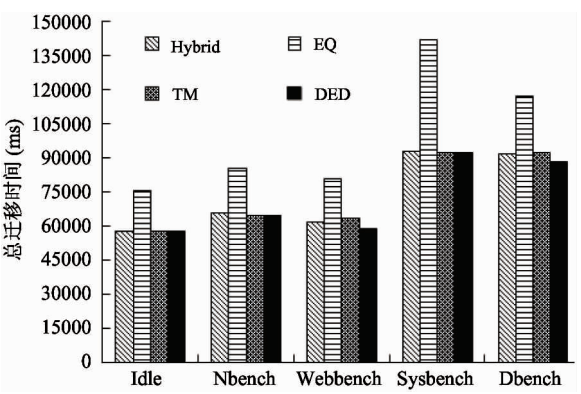

图 12　总迁移时间

另外改变 Sysbench 场景——memory-total-size 参数可以调整内存数据传输量,利用这一特性测量了不同迁移方法对系统和应用性能的影响见图 13,在完成不同数据传输量的任务时,非迁移的状况下用时最短。而迁移的情况下,完成相同的任务量,所用时间有所增加。不同的迁移方法影响的程度也是不同的。

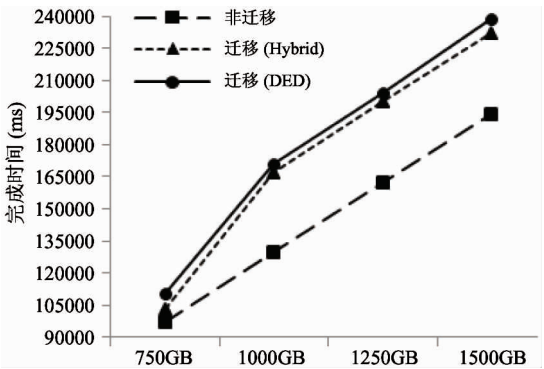

图 13　不同任务完成时间

四种不同的任务下,混合内存复制方式分别损失 6% ,29% ,23% 和 19% 。而等差划分方式下性能损失 14% ,32% ,26% 和 23% ,略高于前者。等差划分的方法比单纯的采用混合内存复制方式造成了更大的系统和应用性能损耗。

4.4　不同内存测试

实际使用的虚拟机,内存大小是不尽相同的,为此增加了 256MB,1024MB 和 2048MB 的内存进行测试。以观察基于等差划分的方法在不同内存中的表现。并选用了 Dbench 场景作为工作负载。图 14 ~ 图 16 给出了试验结果。

对于同一个应用场景,当内存的大小发生变化时,后拷贝方式传送的内存页也发生一定的变化,但两种算法的变化趋势一致,混合内存拷贝方式增加时,等差划分的方式也增加,减少时也相应减少。从 256MB 到 2048MB 内存,依次减小的比例分别为 12% ,11% ,33% 和 43% 。

从上图中可以看出,同步内存位图的时间随着内存的增加而增加,并不是随着第二轮传输的内存页数目的变化而发生改变。两种算法的变化都有增加的趋势。从256MB到2048MB依次减少的比例为

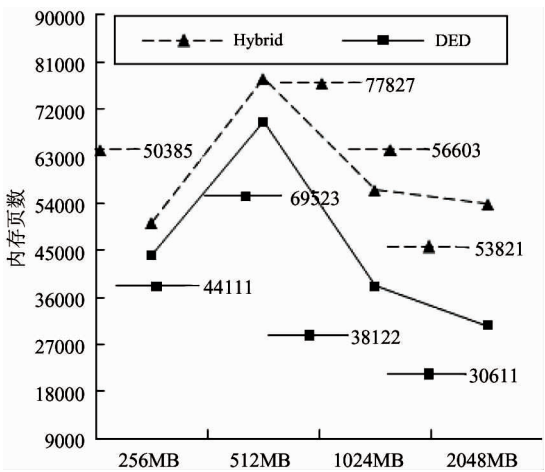

图 14　后拷贝方式内存页数

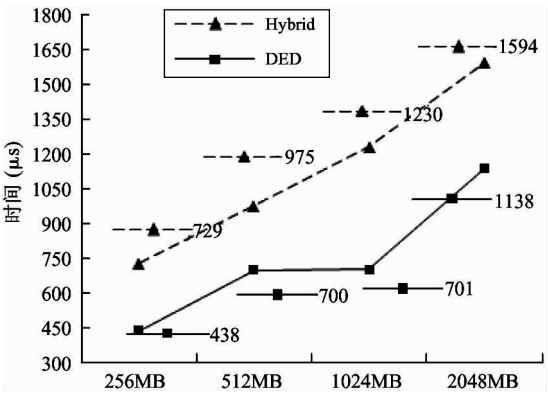

图 15　同步内存位图时间

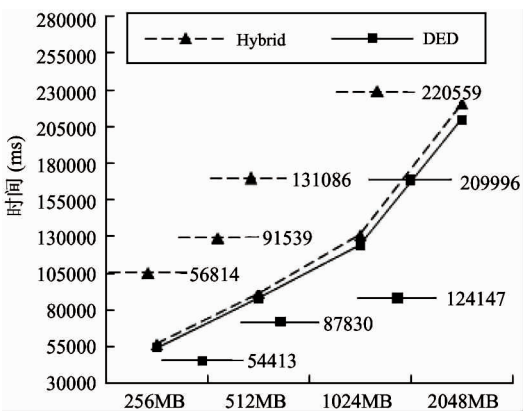

图 16　总迁移时间

40% ,28% ,43% 和 29% 。只有 2048MB 内存时,同步内存位图的时间减少的比列小于第二轮减少的内存页的比例,其余三种情况均大于。这说明传输内存位图时间与设置内存缺页时间的比例随着内存的增加,有增大的趋势。

对于总迁移时间而言,随着内存的增加,两种算法都呈现出上升的趋势,从 256MB 到 2048MB 内存,基于等差划分迁移技术改进的比例依次为 4%,4%,5% 和 5% 呈基本相同的态势。可见第一轮同步全部内存页的时间占总迁移时间的比例很大。

5 结 论

虚拟机实时迁移技术是虚拟化领域的研究热点之一。本文对虚拟机实时迁移中的混合内存方式进行了改进,在同步整个内存页阶段,按照等差数列的形式对整个内存页划分了不同的段,提出了基于等差划分的虚拟机实时迁移技术。同时为了配合等差划分这种技术,又安排了计数排序的预处理阶段,和分批次同步变脏的内存页位图技术。试验结果表明,在不同工作负载的情况下,基于等差划分的虚拟机实时迁移技术可以减少后拷贝阶段传输的内存页的数量,同时带动总迁移时间和同步位图时间的减小。与混合内存方式相比,最好的情况下可以使总迁移时间减少 5% 上下,同时使同步位图时间减少 31% 左右,使后拷贝阶段传输的内存页减少 58%。基于等差划分的虚拟机实时迁移技术适用于不同的内存,对于不同内存下的同一负载,总迁移时间,同步内存位图时间和第二轮传输的内存页数目都有所改进。后续工作中,将着重研究多虚拟机实时迁移中存在的问题,因为在某些环境下有些虚拟机彼此关联,需要同时被迁移。

参考文献

[1] Clark C, Fraser K, Hand S, et al. Live migration of virtual machines. *Networked Systems Design &Implementation (NSDI)*, 2005, 2:273-286

[2] Theimer M M, Lantz K A, Cheriton D R. Preemptable remote execution facilities for the V-system. *ACM Sigops Operating Systems Review*, 1985, 19(5):2 - 12

[3] Jin H, Deng L, Wu S. Live virtual machine migration with adaptive memory compression. In: Proceedings of the IEEE International Conference on Cluster Computing and Workshops, New Orleans, USA, 2009. 1-10

[4] Zhang X, Huo Z G, Ma J, et al. Exploiting data deduplication to accelerate live virtual machine migration. In: Proceedings of the IEEE International Conference on Cluster Computing, Heraklion, Greece, 2010. 88-96

[5] Svard P, Hudzia B, Tordsson J, et al. Evaluation of delta compression techniques for efficient live migration of large virtual machines. *Virtual execution environments (VEE)*, 2011, 46(7):111 - 120

[6] Ma Y Q, Wang H B, Dong J K, et al. ME2: efficient live migration of virtual machine with memory exploration and encoding. In: Proceedings of the IEEE International Conference on Cluster Computing, Beijing, China, 2012. 610-613

[7] Du Y Y, Yu H L, Shi G Y, et al. Microwiper: efficient memory propagation in live migration of virtual machines. In: Proceedings of the 39th International Conference on Parallel Processing, San Diego, USA, 2010. 141-149

[8] Svärd P, Tordsson J, Hudzia B, et al. High performance live migration through dynamic page transfer reordering and compression. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Athens, Greece, 2011. 542-548

[9] Zou Q X, Hao Z Y, Cui X, et al. Counting sort for the live migration of virtual machine. In: Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, USA, 2013. 1-5

[10] Adel A, Kamran Z. Improving the time of live migration virtual machine by optimized algorithm scheduler credit. In: Proceedings of the 4th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 2014. 346-351

[11] Liu Z B, Qu W Y, Liu W J, et al. Xen live migration with slowdown scheduling algorithm. In: Proceedings of the International Conference on Parallel and Distributed Computing, Ap-plications and Technologies (PDCAT), Wuhan, China, 2010. 215-221

[12] Jin H, Gao W, Wu S, et al. Optimizing the live migration of virtual machine by CPU scheduling. *Journal of Network and Computer Applications*. 2011, 34(4):1088-1096

[13] Hines M R, Deshpande U, Gopalan K. Post-copy live migration of virtual machines. *ACM Sigops Operating Systems Review*, 2009, 43(3):14-26

[14] 陈阳,怀进鹏,胡春明. 基于内存混合复制方式的虚拟

机在线迁移机制. 计算机学报,2011, 34(12):2278-2291

[15] Kashyap S, Dhillon J S, Purini S. RLC - A reliable approach to fast and efficient live migration of virtual machines in the clouds. In: Proceedings of the IEEE 7th International Conference on Cloud Computing (CLOUD), Anchorage, USA, 2014. 360-367

[16] Hu L, Zhao J, Xu G C, et al. HMDC: live virtual machine migration based on hybrid memory copy and delta compression. *Applied Mathematics & Information Sciences* (*CLOUD*), 2013,7(2L): 639-646

[17] Sahni S, Varma V. A hybrid approach to live migration of virtual machines. In: Proceedings of the IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India, 2012. 1-5

[18] Liu H K, Jin H, Liao X F, et al. Live migration of virtual machine based on full system trace and replay. In: Proceedings of the 18th ACM international Symposium on High Performance Distributed Computing (HPDC), Garching, Germany, 2009. 101-110

[19] Chanchio K, Thaenkaew P. Time-bound, thread-based live migration of virtual machines. In: Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Chicago, USA, 2014. 364-373

Division of arithmetic progression for virtual machines' live migration

Zou Qingxin[*,**,***], Hao Zhiyu[***], Yun Xiaochun[*,***], Wang Chonghua[**,***]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

([***] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

Abstract

The live migration of virtual machines was studied. The performances of the pre-copy algorithm, the post-copy algorithm and the hybrid-copy algorithm for memory migration in local area networks were analyzed, and then, on the basis of the improvement of the hybrid-copy algorithm, a technique for the live migration of virtual machines based on the division of arithmetic progression was presented. The technique divides the memory pages transmitted in the first round into many segments similar to the form of arithmetic progression, so the memory pages transmitted in the second round can be decreased. The experiments performed under different workloads showed that compared with the hybrid-copyalgorithm, the proposed technique reduced the time of synchronous memory bitmap by 25%, and decreased the pages of post-copy by 29% and the total migration time by 2.2% on average, improving and decreased the performance of virtual machines' live migrationto some extent.

Key words: live migration, virtual machine, pre-copy, post-copy, hybrid-copy, division of arithmetic progression