

量的维度进行考虑,最终得出虚拟机的优化放置方案.这一方法基于 K -Means 算法,最终目标是将集群中的所有虚拟机聚成 X 类, X 为集群中的 Hypervisor 的数量.集群中每台虚拟机都会由一个独立的特征向量进行刻画.

在生成特征向量时,我们需要对监测模块收集到的数据进行预处理操作.如前文所述,我们监测到了虚拟机集群中的应用边界以及应用中各虚拟机的关联.这天然决定了某些虚拟机的特征向量属于同一组,这些虚拟机之间具有一定的相似性,且对放置有类似的倾向.此外,对监测到的 I/O 性能数据和 SSD 缓存使用状况的数据需要进行归一化操作,确保最终聚类结果在满足 SSD 服务能力约束的情况下,最大限度地使用 SSD 资源.

我们使用如下维度来构成特征向量,这些特征值都归一化到 0~1 的数据范围内.这里,我们以 Hadoop 为例解释这些特征的具体表现.

- (1) 虚拟机 I/O 性能:主要包括虚拟机的 CPU 处理时间中的 I/O 时间、磁盘读写带宽和读写 IOPS.这些指标反映了虚拟机在应对当前工作负载时的 I/O 压力.例如,对于 Hadoop 中的 HDFS 数据节点而言,其平均 I/O 时间相对较高,且集中在顺序读写操作;对于 HDFS 元数据节点而言,I/O 负载主要集中在随机读写操作.这一特性可以区分不同 I/O 访问频率的虚拟机.
- (2) SSD 缓存特性:主要包括缓存容量、缓存利用率、读写操作数和读写命中率.这些指标反映了虚拟机对缓存的使用情况.其中,缓存利用率和缓存命中率直观反映了虚拟机对缓存的依赖程度和缓存的性价比.SSD 对于随机读写的提升比顺序读写更为明显.以 Hadoop 为例,HDFS 的数据节点的缓存读写命中率会比 HDFS 元数据节点要低,同时占用更多的缓存空间.这一特性可以区分不同访问模式的虚拟机.此外,缓存容量是由 SSD 缓存系统中的容量规划组件根据虚拟机的 I/O 特征和应用的行为确定的,不在本文的讨论范围内.
- (3) 网络特征:主要包括虚拟机对网络带宽的使用情况,包括与从属应用相关的连接数以及这些连接占用的网络带宽.这些指标反映了虚拟机对应用内其他虚拟机的依赖程度,间接反映了虚拟机的内聚倾向.对 Hadoop 应用而言,计算节点(YARN)到存储节点(HDFS)的网络带宽较高,建立的连接数也相对较多,这说明计算节点对存储节点存在一定的依赖.这一特性可以区分不同内聚倾向的虚拟机.
- (4) 虚拟机的内聚程度:主要由虚拟机的网络特征和所依赖目标虚拟机的 I/O 性能综合计算得出.在计算虚拟机内聚程度时,我们将 0~1 的数据范围等分为 X 份(X 为 Hypervisor 数量),并将每个应用中所有虚拟机按照对应的内聚关系分组,最后将这些分组中的虚拟机的内聚程度放置在特定的 X 等分数据范围内,并在数据中心点周围随机取值.内聚程度越高的虚拟机组,这一值的方差越小,即偏移的随机量越小.对 Hadoop 应用而言,对特定存储节点依赖程度较高的计算节点到这一存储节点的偏差相对较小,而与不相关的存储节点偏差较大,从而体现虚拟机的内聚倾向.
- (5) 虚拟机的隔离程度:这一指标主要基于先验知识.与处理虚拟机的内聚程度类似,我们将 0~1 的数据范围等分为 X 份(X 为 Hypervisor 数量),之后以应用为单位,将需要隔离的虚拟机组(如 Hadoop 集群中的元数据服务器和备份元数据服务器、ZooKeeper 中的 Quorum)分到不同的数据范围内,并在对应的数据中心点周围随机取值.对于其他类型的应用,使用第 3.2 节所述的默认策略确定隔离程度.

在执行具体的聚类算法时,我们做了如下改进.

- (1) 在选取初始质心时,我们并非随机选取,而是选取了所有应用中虚拟机隔离程度最高的最多 X 台虚拟机(X 为 Hypervisor 数量),这一初始质心选取方案将会减少质心的调整次数,从而帮助算法快速收敛到最终结果.
- (2) 在进行迭代的聚类操作时,我们额外考察了每个聚类中的虚拟机的 I/O 负载以及对缓存容量、SSD 带宽以及 IOPS 的需求,以满足 SSD 缓存供给能力的约束.这一改进确保最终生成的虚拟机放置方案不会导致虚拟机对 SSD 缓存资源的争用.
- (3) 聚类算法的具体参数中包含对每个聚类的元素上下限的设置,控制每个聚类中包含的元素个数,从而防止降低数据倾斜的概率.

最终,聚类算法会生成优化的虚拟机放置方案.需要注意的是,在进行特征选取和聚类算法计算时,我们并

没有考虑虚拟机之间的网络延迟,这是由于在虚拟化环境下,物理服务器通常位于同一个数据中心,物理服务器间以及虚拟机间的网络延迟相比迁移的开销而言可以忽略.

3.4 迁移方案生成

虚拟机的动态迁移是代价相对较高的操作,特别是在 SSD 缓存系统中,需要在进行动态迁移时移动缓存中的脏数据,对性能的影响更为明显.因此,在通过聚类算法得到优化的虚拟机放置方案之后,我们需要计算出最少的迁移步骤以及优化的迁移顺序,尽可能地降低迁移操作对运行时虚拟机的影响.

- 首先,聚类算法的结果并不包含聚类到主机的映射,我们需要根据当前 Hypervisor 上的虚拟机放置情况找出 Hypervisor 到聚类的最大匹配,从而移动最少数量的虚拟机即可转变到最优放置方案.
- 其次,在得到 Hypervisor 到聚类的映射的基础上,我们需要匹配虚拟机的迁移操作并最终给出迁移操作的序列.
- 最后,我们需要根据先验知识确定开销最小的虚拟机迁移顺序.

Hypervisor 集群上当前所有虚拟机的放置状态 P_0 可以表达为一个集合:

$$P_0 = \{H_1, H_2, \dots, H_n\},$$

其中,

$$H_i = \{VM_{i_1}, VM_{i_2}, \dots, VM_{i_k}\}$$

P_0 集合中,元素 H_i 代表不同的 Hypervisor, VM_{i_k} 代表在 H_i 上放置的虚拟机.

聚类结果 C 也可以表达为一个集合:

$$C = \{c_1, c_2, \dots, c_n\},$$

其中,

$$c_i = \{VM_{i_1}, VM_{i_2}, \dots, VM_{i_k}\}.$$

C 集合中的元素 c_i 代表不同的聚类, VM_{i_k} 代表属于聚类 c_i 的虚拟机.

我们的目标是找到一个 H_i 到 c_i 的映射,使得移动最少的虚拟机即可将放置状态 P 变为理想放置状态.

查找最优映射的方法 CalculateOptimizedMapping 如图 5 所示,其主要原理是查找所有映射可能的映射方案并计算其中虚拟机的匹配度,其算法复杂度为 $O(nm)$,其中, n 为聚类集合数量, m 为集合中元素个数.其中使用到的函数 FindMapping 查找所有可能映射方案,即找到主机到聚类的全排列方案.考虑到 Hypervisor 个数相对较少,本文中 FindMapping 函数直接使用了递归求解,其算法复杂度为 $O(n!)$.函数 CalculteFitness 用于计算特定映射方案的匹配度,其算法复杂度为 $O(n^2)$.最终, CalculateOptimizedMapping 遍历所有可能的匹配并返回最优匹配方案.

```

算法 1. CalculateOptimizedMapping.
输入: 当前虚拟机放置方案  $P_0$ ; 聚类结果  $C$ .
输出: 物理机到聚类的映射  $M$ .
mappings ← FindMapping( $P_0, C$ );
currentMaxFitness ← 0;
foreach mapping  $m$  in mappings do
    fitness ← CalculateFitness( $m$ );
    if fitness > currentMaxFitness then
        currentMaxFitness ← fitness;
         $M$  ←  $m$ ;
    end
end
return  $M$ ;

```

Fig.5 Calculate optimized mapping of hypervisor and cluster

图 5 计算的主机到聚类的最优映射

计算特定映射的虚拟机匹配度的算法如图 6 所示.计算匹配度的过程是对放置方案中的每一个 Hypervisor 上的虚拟机集合,计算其对应的聚类中的虚拟机集合的交集的秩.映射的虚拟机匹配度即为所有对应交集的秩的总和,其算法复杂度为 $O(n^2)$.

在计算完 Hypervisor 到聚类的最优映射之后,我们需要计算从当前虚拟机放置方案到目标聚类的虚拟机

迁移操作集合.

在这里,我们定义两个原子操作.

- (1) 迁入虚拟机: $H_i.In(VM_j)$,代表将虚拟机 VM_j 从其他 Hypervisor 迁入 H_i .
- (2) 迁出虚拟机: $H_i.Out(VM_j)$,代表需要从 Hypervisor H_i 中迁出虚拟机 VM_j .

```

算法 2. CalculateFitness.
输入:物理机到聚类的映射  $M$ .
输出:虚拟机匹配度  $f$ .
 $f \leftarrow 0$ ;
foreach  $VMSet\ s_0$  in  $M.host$  do
     $VMSet\ s_1 \leftarrow M.cluster$ 
     $f \leftarrow f + count(intersect(s_0, s_1))$ ;
end
return  $f$ ;

```

Fig.6 Calculate fitness of the mapping

图 6 计算映射的匹配度

计算迁移操作的算法如图 7 所示,其主要原理是:比较原始放置方案和目标聚类结果中对应的虚拟机集合,需要迁出的虚拟机即为原始虚拟机集合与交集的差集;需要迁入的虚拟机即为聚类结果中的虚拟机集合与交集的差集.

```

算法 3. CalculateMigrationOperations.
输入:物理机到聚类的映射  $M$ .
输出:迁移原子操作集合  $OpSet$ .
 $OpSet \leftarrow Empty$ ;
foreach  $VMSet\ s_0$  in  $M.host$  do
     $VMSet\ s_1 \leftarrow M.cluster$ 
     $Set\ outSet \leftarrow s_0 - intersect(s_0, s_1)$ ;
     $Set\ inSet \leftarrow s_1 - intersect(s_0, s_1)$ ;
    foreach  $VM\ vm$  in  $outSet$  do
         $OpSet.add(M.host.Out(vm))$ ;
    end
    foreach  $VM\ vm$  in  $inSet$  do
         $OpSet.add(M.host.In(vm))$ ;
    end
end
return  $OpSet$ ;

```

Fig.7 Calculate migration operations

图 7 计算迁移操作

之后,我们匹配对应的迁移原子操作以生成迁移方案.算法如图 8 所示,这一算法匹配了迁移原子操作集合中从属于同一个虚拟机的迁入和迁出操作.

最后,我们将根据先验规则,配合监测到的虚拟机的 I/O 负载以及虚拟机在应用中的交互,最终确定迁移顺序.基本原则如下.

- (1) 针对虚拟机持续监测,优先迁移 I/O 负载较低的虚拟机.较低的 I/O 负载意味着虚拟机目前处于相对空闲的状态,执行动态迁移所带来的性能影响会相对较小.
- (2) 针对虚拟机持续监测,优先迁移对缓存依赖程度较低的虚拟机.这些虚拟机的缓存脏数据较少,可以以相对较快的速度完成动态迁移操作.
- (3) 针对应用持续监测,优先迁移被依赖程度较低的虚拟机.被依赖程度较低意味着对虚拟机有数据本地性需求的其他虚拟机的数量较小,迁移这一虚拟机并不会对其他虚拟机的性能产生较为严重的影响.
- (4) 持续监测网络带宽用量,限制虚拟机动态迁移的带宽.这一规则限制了虚拟机动态迁移的速度,以牺牲迁移效率为代价降低对性能的影响.考虑到应用的工作负载是长期的、具有周期性的,低开销水平

的动态迁移是可以接受的.

最终生成的优化迁移顺序会交由具体的虚拟机动态迁移模块完成迁移操作.在迁移虚拟机时,会持续监测虚拟机的性能和当前负载,在虚拟机负载较低时执行迁移,从而降低迁移操作对虚拟机性能的影响.

```

算法 4. MatchMigrationOperations.
输入:迁移原子操作集合 OpSet.
输出:迁移方案 Plan.
Plan←Empty;
foreach AtomOperation op in OpSet do
    if op.visited==false && op.type==Out then
        foreach AtomOperation anotherOp in OpSet do
            if (op!=anotherOp)
                && (anotherOp.type==In)
                && (op.VM==anotherOp.VM) then
                    matchedOp←anotherOp;
                    break;
            end
        end
        Plan.add(op.VM to matchedOp.VM);
        op.visited←true;
        matchedOp.visited←true;
    end
end
return Plan;
    
```

Fig.8 Match migration operations

图 8 匹配迁移操作

3.5 系统实现

我们在 Xen Hypervisor^[16]上实现了虚拟化环境下面向多目标优化的自适应 SSD 缓存原型系统.系统架构如图 9 所示.

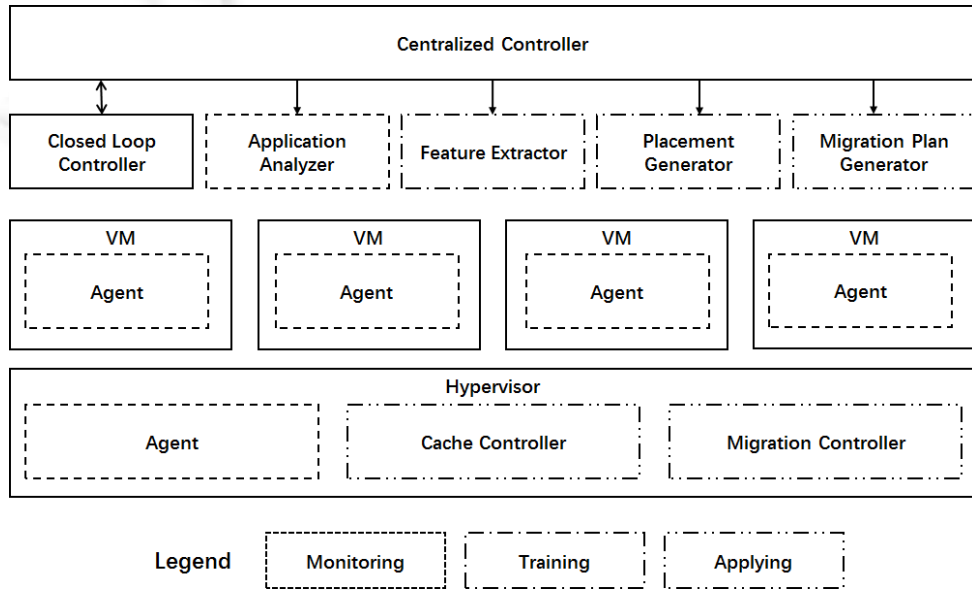


Fig.9 System architecture

图 9 系统架构

按照自适应闭环中的角色来划分,系统中的主要功能模块如下.

- 闭环控制部分
 - (1) 中央控制器:中央控制器用于协调整个自适应闭环的执行.中央控制器开放了 RESTful 风格的 API 供管理员跟踪系统的执行状况,并支持与虚拟化管理程序的深度整合.
 - (2) 闭环控制器:闭环控制器持续调用监测模块,基于滑动窗口机制,比较上一个窗口和当前窗口中持续监测的虚拟机数据的变化情况确定是否触发新一轮的调整操作.当感知到虚拟机集群规模发生变化、应用的边界或应用内虚拟机之间的关联发生变化,或是通过先验知识初步得出的应用的虚拟机放置倾向性发生变化时,会开始新一轮的闭环执行.
- 监测部分
 - (1) 部署在虚拟机上的代理(agent)模块:部署在虚拟机上的代理模块通过串口和 Xen Hypervisor 进行通信.Xen Hypervisor 会将虚拟机中的串口映射到 Hypervisor 的特定 Socket 上,从而允许通过 TCP 协议进行交互.代理模块开放了专有协议,允许其他组件执行具体的监测和维护指令来获得虚拟机内部的性能信息,主要包括 CPU、内存、磁盘 I/O 以及网络 I/O 的使用情况.此外,部署在虚拟机上的代理模块也会执行相应的维护命令来确定虚拟机在应用中的角色、虚拟机与应用中其他虚拟机的交互,并对应用的执行日志进行分析,以感知应用对虚拟机放置的倾向性.
 - (2) 部署在 Hypervisor 上的代理(agent)模块:由于 SSD 部署在每个 Hypervisor 上,SSD 缓存本身是对虚拟机透明的,无法在虚拟机内部监测到,需要由部署在 Hypervisor 上的代理模块处理.这一模块主要用于监测 SSD 缓存状态,包括缓存大小、缓存利用率、读写操作数以及读写命中率等.
 - (3) 应用探查模块:应用探查模块主要用于控制部署在虚拟机和 Hypervisor 上的代理模块,收集应用相关的性能数据、虚拟机在应用中的角色以及虚拟机之间的交互,结合从代理模块中获得的虚拟机 I/O 性能和 SSD 缓存状态数据,最终确定虚拟机集群中的应用边界以及应用对虚拟机放置的倾向性,为聚类算法的运行提供支持.
- 训练部分
 - (1) 特征抽取模块:特征抽取模块主要收集监测模块传递的虚拟机 I/O 性能数据和 SSD 缓存状态数据,以及应用探查模块传递的应用状态以及对虚拟机放置的倾向性数据,并以虚拟机为单位将其加工成特征向量,供聚类算法调用.
 - (2) 放置方案生成模块:从特征抽取模块中获得所有虚拟机的特征向量,并执行 *K*-Means 聚类算法,生成具体的放置方案.
- 执行部分
 - (1) 迁移方案生成模块:调用 Xen Hypervisor 提供的 API 获得集群中所有 Hypervisor 上部署的虚拟机,得到当前的虚拟机放置状态;之后,根据前文所述算法生成具体的虚拟机迁移操作集合,并按照先验规则确定迁移的优先级.
 - (2) 缓存控制器:缓存控制器部署在每个 Hypervisor 上,用于执行实际的缓存分配操作,并在虚拟机动态迁移过程中提供缓存迁移支持,包括缓存脏数据管理以及缓存设备的添加和删除操作,保证虚拟机在完成动态迁移操作后缓存立即可用.
 - (3) 动态迁移模块:部署在每个 Hypervisor 上,维护 Hypervisor 上的虚拟机状态,并执行具体的虚拟机动态迁移操作.

4 实验分析

我们在两台 Hypervisor 上进行实验,Hypervisor 的硬件配置见表 1.我们部署了一个由 6 台虚拟机组成的集群,每台虚拟机使用 2 个虚拟 CPU(vCPU)和 2GB 内存,并配置了基于 SSD 缓存的存储.考虑到本节中的实验场景,我们为每台虚拟机分配了 16GB 的 HDD,配以 256MB 的 SSD 缓存.

我们选取了 I/O 基准测试、Hadoop 应用和 ZooKeeper 应用这 3 个场景来验证我们的原型系统的有效性.

在这 3 个实验场景中,CPU、内存、网络带宽和 HDD 空间都不存在瓶颈,唯一的瓶颈是 SSD 缓存.在进行实验时,将我们的方法与随机迁移方法进行了对比.随机迁移方法作为基准的对照算法,并不会感知虚拟机的状态,其主要操作流程是每经过固定的时间间隔就随机选择集群中的一台虚拟机,并将其动态迁移至另一个 Hypervisor 上,并同时迁移 SSD 缓存.对于 I/O 基准测试、Hadoop 应用以及 ZooKeeper 应用这 3 个场景均进行了 5 组测试并选取了平均值,以尽可能地消除随机算法带来的偶然性,同时保证对照组的可信度.

Table 1 Hardware configuration of Hypervisors

表 1 Hypervisor 硬件配置

项目	描述
CPU	Intel Xeon CPU E5-2620
内存	32GB
SSD	Intel SSD 535 240GB
共享存储	通过 iSCSI 连接到后端分布式存储

考虑到虚拟机动态迁移方法的基本目标是保证所关注的资源尽量不产生争用,在我们的实验场景中,现有的以 CPU、内存、网络带宽和 HDD 空间作为目标的虚拟机动态迁移方法与随机迁移方法会有类似的表现;相比之下,去除了异常数据点的随机迁移方法涵盖了更多情形,更适合作为对照方法.

在实验过程中,我们动态地切换了虚拟机集群上的应用类型,并进一步切换了 Hadoop 的工作负载,自动触发了自适应闭环的执行,计算新的优化迁移方案,并自动触发了虚拟机的动态迁移操作.下文所述的 I/O 基准测试以及针对大数据处理应用和分布式协同应用的测试的实验结果是迁移过程稳定之后的时间片段内的结果.为公平起见,我们将随机迁移的时间间隔以及闭环控制器的时间窗口设置为 1 分钟.

4.1 I/O基准测试

首先,我们选取了 I/O 基准测试 fio,用来验证我们的 SSD 缓存系统是否能够自适应地感知 I/O 负载并实现 SSD 缓存资源的负载均衡.为模拟极端的随机读写测试场景,我们使用 fio 以 32KB 块大小在每台虚拟机上并发测试随机读写 1GB 大小文件,并限制虚拟机对内存缓冲区的使用,以最大限度地使用 SSD 缓存的供给能力.如前文所述,我们为每个虚拟机分配了 256MB 缓存,远小于测试场景中 1GB 的总工作集大小,这也可以充分触发缓存置换和数据同步操作,模拟实际的 I/O 密集型负载场景.

随机读测试的实验结果如图 10 所示,随机写测试的实验结果如图 11 所示.图中横轴表示虚拟机节点的编号,柱状图的纵轴表征了各节点上的 IOPS.折线图的纵轴表征了与随机迁移虚拟机相比,我们的方法在 IOPS 性能上的比值.

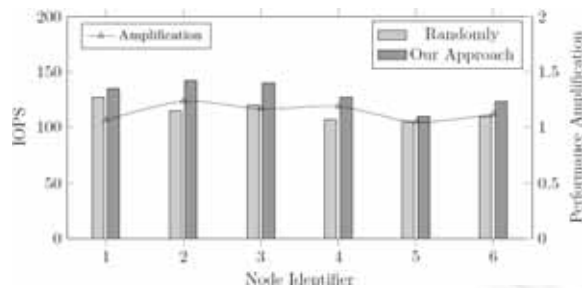


Fig.10 Random read performance

图 10 随机读测试的性能

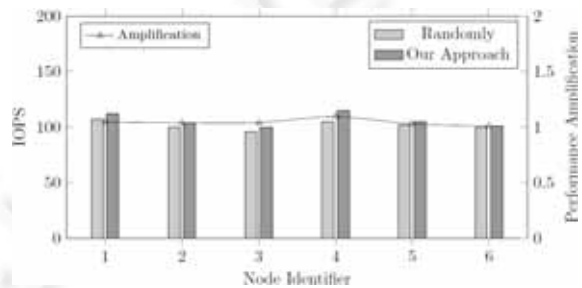


Fig.11 Random write performance

图 11 随机写测试的性能

可以看出,与随机迁移虚拟机相比,我们的迁移方案考虑到虚拟机的 I/O 负载,将高负载的虚拟机迁移到了不同的 Hypervisor,从而在随机读测试的 IOPS 方面达到了平均 15% 的性能提升;在随机写测试方面,受限于缓存容量以及缓存策略,会频繁触发与后端分布式存储系统的同步操作,因此在这一场景下,只能带来平均 6% 的性能提升.

4.2 大数据处理应用的性能

我们首先选取了 Hadoop 自带的一系列微基准测试(Micro benchmarks)来评价我们的迁移方法在应对 Hadoop 任务时的性能表现.我们选取了一组 CPU 敏感型的应用 PI-10 和 PI-20,运行 Hadoop 自带的 pi 基准测试,即使用 Quasi-Monte Carlo 方法计算圆周率.PI-10 代表采用了 10 个 Map 任务,PI-20 代表采用了 20 个 Map 任务.我们同时选取了一组 I/O 敏感型应用:顺序写(write)、顺序读(read)、随机读(randr)、反向读(backr),分别取自 Hadoop 自带的 TestDFSIO 基准测试.

评价 Hadoop 任务执行性能的主要指标是执行时间,实验结果如图 12 所示.图中横轴代表 Hadoop 的微基准测试的类型,柱状图纵轴表征了测试的执行时间,折线图纵轴表示了我们的方法相对于随机迁移而言在执行时间方面的减少比例.PI-10 和 PI-20 的实验结果表明:我们的方法在应对 CPU 敏感型应用场景时,性能与随机迁移类似;对于 I/O 敏感型应用而言,我们的方法考虑到的是 Hadoop 的数据本地性,因此在执行时间方面比随机迁移方法平均降低了 25%.

此外,我们还评价了 I/O 敏感型应用的吞吐率,实验结果如图 13 所示.柱状图的纵轴表征了执行对应测试的吞吐率,折线图纵轴表征了我们的方法与随机迁移方法相比吞吐率的比值.值得注意的是,这一吞吐率实际表征的是在测试执行过程中总共需要操作的数据量与实际操作数据时间的比值,实际操作数据的时间会远小于执行时间.从吞吐率角度看,我们的方法比随机迁移平均提升了 39%,对读操作和随机读操作提升更为明显.

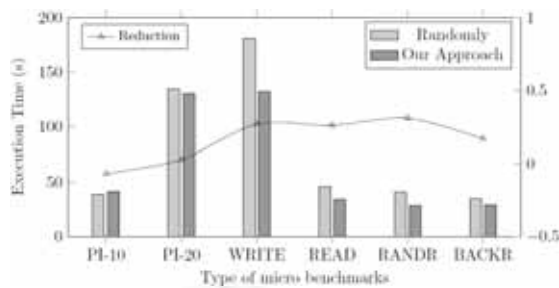


Fig.12 Execution time of Hadoop benchmarks

图 12 Hadoop 基准测试的执行时间

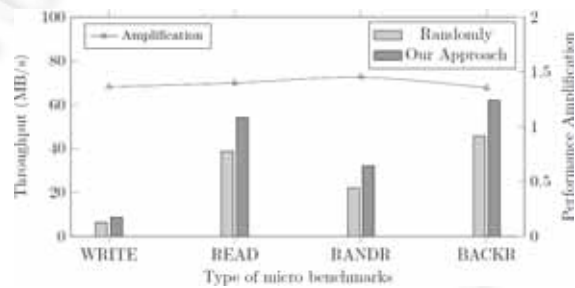


Fig.13 Throughput of Hadoop I/O benchmarks

图 13 Hadoop I/O 基准测试的吞吐率

4.3 分布式协同应用的可靠性

我们采用了逻辑距离这一简单的指标来表征分布式协同应用 ZooKeeper 的可靠性.我们定义:当两台虚拟机位于同一 Hypervisor 上时逻辑距离为 0,当位于不同 Hypervisor 上时逻辑距离为 1.集群的逻辑距离为集群中所有节点两两之间逻辑距离之和.当某一 Hypervisor 出现单点失效时,会导致位于这一 Hypervisor 上所有的虚拟机(即逻辑距离为 0 的虚拟机)宕机,而对其他虚拟机没有影响.因此,集群的逻辑距离越大,可以间接推断出集群本身的可靠性越高.

我们比较了随机迁移方法和我们的方法,在达到相对稳定状态时,其逻辑距离分别为 5 和 9.我们的虚拟机迁移方法最终将虚拟机平均分布在 2 台 Hypervisor 上,从而保证在 Hypervisor 出现单点失效时,受影响的虚拟机数量最小.

我们还比较了 ZooKeeper 应用的性能.我们选取了 ZooKeeper 的作者 Hunt 开发的 ZooKeeper Smoketest^[17],主要用于评价 ZooKeeper 主要操作的延迟.我们测试了访问 ZooKeeper 集群中所有 6 个虚拟机节点时的异步操作速度,实验结果如图 14 所示.图中横轴表征了操作类型,主要包括创建永久(permanent)节点(CREATE)、写入数据(SET)、读取数据(GET)、删除永久节点(DELETE)、观察节点(WATCH).图中柱状图纵轴表征了操作速度,数值为每秒可执行的操作数,折线图纵轴表征了我们的虚拟机迁移方法与随机迁移相比在操作速度上的比例.实验结果表明:我们的虚拟机迁移方案相比随机迁移方案具有类似的性能,平均性能差距为 5%.这一性能差距,在保证 ZooKeeper 可靠性的前提下是可以接受的.

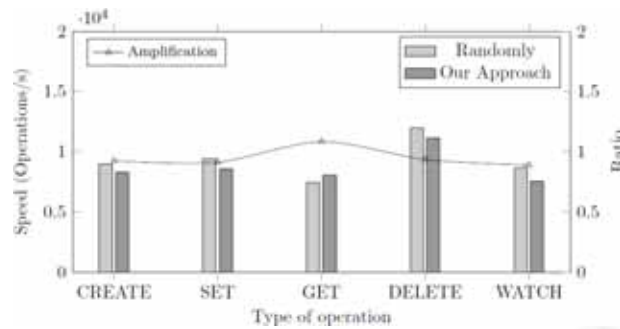


Fig.14 Performance of ZooKeeper—Async operations

图 14 ZooKeeper 异步操作性能

4.4 迁移开销

在实验过程中,我们持续测量了虚拟机动态迁移的时间以及动态迁移执行过程中的性能干扰.我们测量到单台虚拟机的动态迁移时间在 10s~15s 之间,对网络带宽的占用平均为 40%.我们测量了在虚拟机动态迁移过程中执行的 Hadoop 计算密集型 and I/O 密集型应用的任务执行时间.受益于优化的迁移顺序以及对迁移速度的控制,对 Hadoop 任务的执行时间的影响平均为 7%.这一额外开销在大规模长时间的工作负载的场景下是可以接受的.

5 相关工作

现有的缓存资源管理的相关工作主要从缓存本身出发,以缓存的特点及相关关键指标驱动缓存资源管理,最终实现性能提升或降低后端存储系统负载的目的.如在 SSD 和 HDD 构成的混合存储系统中考虑 SSD 应对不同类型 I/O 负载的差异性,进行选择性的数据缓存^[18]以提升性能;或使用本地磁盘介质作为网络存储系统的缓存^[19],以降低网络存储系统的负载,提升系统性能.

许多研究工作考虑 SSD 缓存本身的特点进行缓存分配.RAF^[20]引入成本收益模型,以缓存对 SSD 而言价值较高的随机读写数据,降低 SSD 的响应时间,并提升寿命.Centaur^[6]将 Hypervisor 上的 SSD 缓存进行分区并分配给不同的虚拟机,通过各虚拟机的缓存大小,以满足诸如缓存 Miss 率或虚拟机 I/O 响应时间等具体性能指标的优化需求.S-CAVE^[9]基于缓存命中率和虚拟机频繁使用的缓存空间大小导出了虚拟机对缓存空间的需求,并以此实现 SSD 缓存资源分配.vCacheShare^[21]从虚拟机的 I/O 行为和特点出发,结合虚拟机的数据本地性以及缓存命中率和重用程度等特性,指导 SSD 缓存资源分配.这些工作都是以 SSD 缓存的关键指标作为调整依据,面向单台 Hypervisor 的场景,并没有从全局角度考虑 SSD 缓存资源的供给能力,也难以在 SSD 缓存出现资源争用时进行有效应对.

一些研究工作考虑了缓存资源出现争用的情况并加以解决.CloudCache^[8]提出了一种基于 Reused Working Set 的 SSD 缓存方法,可以在缓存命中率相当的前提下有效降低需要缓存的数据大小,从而提高缓存的利用率.此外,CloudCache 也提出了一种以缓存容量为导向的虚拟机动态迁移策略,在虚拟机对缓存容量的需求超过 SSD 总容量时触发虚拟机动态迁移.然而,CloudCache 没有考虑到 SSD 作为存储介质本身对带宽和 IOPS 能力的约束,也没有考虑到应用对虚拟机放置的倾向性,因此,可能在满足缓存容量约束的前提下出现虚拟机对 SSD 的带宽和 IOPS 的争用,也可能在执行迁移的过程中对虚拟机进行了错误的放置,影响了虚拟机的性能或虚拟机上承载应用的可靠性.

此外,一些研究工作考虑到了应用内的工作负载,并以此指导了 SSD 缓存分配.Capo^[22]面向虚拟桌面的场景,考虑到了用户虚拟机中的不同目录和文件的使用模式和对缓存的不同需求.Capo 为用户磁盘内的不同目录设置了不同的缓存策略,以此提高缓存利用率,避免不必要的缓存操作,降低存储系统的负载.文献[23]构建了一个原型文件系统,在底层对 I/O 请求进行分类,针对不同类型的 I/O 请求设置不同的缓存策略,从而实现缓存命中

率和虚拟机性能的提升.这些研究工作对虚拟机内运行的工作负载进行了建模,从而可以有效地进行应对,但它们未能从全局的角度考虑多台虚拟机组成的应用以及应用内部虚拟机关联对缓存资源管理的影响.

6 结论与未来的工作

本文提出了虚拟化环境下面向多目标优化的自适应 SSD 缓存系统,构建了一个自适应闭环,通过持续监测并感知模式变化以驱动虚拟机的动态迁移,最终在容量规划之外实现了对 SSD 缓存利用率、应用性能和应用可靠性的多目标优化.系统在监测时重点关注了虚拟机和 SSD 缓存的性能指标,并感知虚拟机上部署的应用的边界、应用内虚拟机的关联以及应用本身对虚拟机放置的倾向性;之后,使用聚类算法获得优化放置方案,并给出了合理的迁移顺序以降低开销.

这一工作的主要不足之处在于,对应用类型的覆盖范围较小,目前只完整支持 Hadoop 和 ZooKeeper 两类应用,且依赖于一定的先验知识来获得应用对虚拟机放置的倾向.此外,系统的监测粒度也相对有限,未能深入大数据处理应用内部去感知用户代码的行为,以此指导 SSD 缓存负载均衡和对应用性能和可靠性的优化.

未来我们将刻画应用行为以及使用 SSD 缓存时的共性,从这两个角度对应用进行自动分类,更合理地感知应用对 SSD 的需求,从而支持虚拟化环境下更多类型的应用.此外,还将实现更深层次的应用感知,从用户代码层面考虑应用的行为和对 SSD 缓存的使用情况,从而支持更灵活的 SSD 缓存资源管理.

References:

- [1] Gulati A, Shanmuganathan G, Zhang X, Varman P. Demand based hierarchical QoS using storage resource pools. In: Proc. of the 2012 USENIX Conf. on Annual Technical Conf. Boston: USENIX Association, 2012. 1–13.
- [2] Hansen JG, Jul E. Lithium: Virtual machine storage for the cloud. In: Proc. of the 1st ACM Symp. on Cloud Computing. Indianapolis: ACM Press, 2010. 15–26. [doi: 10.1145/1807128.1807134]
- [3] Ye L, Lu G, Kumar S, Gniady C, Hartman JH. Energy-Efficient storage in virtual machine environments. In: Proc. of the 6th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. Pittsburgh: ACM Press, 2010. 75–84.
- [4] Lu L, Pillai TS, Arpaci-Dusseau AC, Arpaci-Dusseau RH. WiscKey: Separating keys from values in SSD-conscious storage. In: Proc. of the 14th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2016. 133–148. [doi: 10.1145/3033273]
- [5] Kim J, Lee D, Noh S H. Towards SLO complying SSDs through OPS isolation. In: Proc. of the 13th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2015. 183–189.
- [6] Koller R, Mashizadeh AJ, Rangaswami R. Centaur: Host-Side SSD caching for storage performance control. In: Proc. of the 2015 IEEE Int'l Conf. on Autonomic Computing. Wurzburg: IEEE Computer Society, 2015. 51–60. [doi: 10.1109/ICAC.2015.44]
- [7] Oh Y, Lee E, Hyun C, Choi J, Lee D, Noh S H. Enabling cost-effective flash based caching with an array of commodity SSDs. In: Proc. of the 16th Annual Middleware Conf. Vancouver: ACM Press, 2015. 63–74. [doi: 10.1145/2814576.2814814]
- [8] Arteaga D, Cabrera J, Xu J, Zhao M. CloudCache: On-Demand flash cache management for cloud computing. In: Proc. of the 14th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2016. 355–369.
- [9] Luo T, Ma S, Lee R, Zhang X, Liu D, Zhou L. S-CAVE: Effective SSD caching to improve virtual machine storage performance. In: Proc. of the 22nd Int'l Conf. on Parallel Architectures and Compilation Techniques. Edinburgh: IEEE Computer Society, 2013. 103–112. [doi: 10.1109/PACT.2013.6618808]
- [10] Shvachko K, Hairong K, Radia S, Chansler R. The hadoop distributed file system. In: Proc. of 2010 IEEE the 26th Symp. on Mass Storage Systems and Technologies (MSST). Incline Village: IEEE Computer Society, 2010. 1–10. [doi: 10.1109/MSST.2010.5496972]
- [11] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E. Apache hadoop YARN: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. Santa Clara: ACM Press, 2013. 1–16. [doi: 10.1145/2523616.2523633]
- [12] Junqueira FP, Reed BC, Serafini M. Zab: High-Performance broadcast for primary-backup systems. In: Proc. of the 2011 IEEE/IFIP 41st Int'l Conf. on Dependable Systems&Networks. Hong Kong: IEEE Computer Society, 2011. 245–256. [doi: 10.1109/DSN.2011.5958223]

- [13] Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: Wait-Free coordination for Internet-scale systems. In: Proc. of the 2010 USENIX Conf. on USENIX Annual Technical Conf. Boston: USENIX Association, 2010. 11–24.
- [14] Intel. Intel solid state drive 750 series. 2016. <http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-750-spec.pdf>
- [15] WD. WD black PC HD series specification sheet. 2016. https://www.wdc.com/content/dam/wdc/website/downloadable_assets/eng/spec_data_sheet/2879-771434.pdf
- [16] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proc. of the 19th ACM Symp. on Operating Systems Principles. Bolton Landing: ACM Press, 2003. 164–177. [doi: 10.1145/945445.945462]
- [17] Hunt P. ZooKeeper smoketest. 2016. <https://github.com/phunt/zk-smoketest>
- [18] Yang PY, Jin PQ, Yue LH. A time-sensitive and efficient hybrid storage model involving SSD and HDD. Chinese Journal of Computers, 2012,35(11):2294–2305 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2012.02294]
- [19] Yin Y, Liu ZJ, Xu L. Cache system based on disk media for network storage. Ruan Jian Xue Bao/Journal of Software, 2009,20(10):2752–65 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/3427.htm> [doi: 10.3724/SP.J.1001.2009.03427]
- [20] Liu Y. Research on caching and prefetching technologies for hierarchical hybrid storage systems [Ph.D. Thesis]. Wuhan: Huazhong University of Science and Technology, 2013 (in Chinese with English abstract). [doi: 10.7666/d.D608789]
- [21] Meng F, Zhou L, Ma X, Uttamchandani S, Liu D. vCacheShare: Automated server flash cache space management in a virtualization environment. In: Proc. of the 2014 USENIX Conf. on USENIX Annual Technical Conf. Philadelphia: USENIX Association, 2014. 133–144.
- [22] Shamma M, Meyer DT, Wires J, Ivanova M, Hutchinson NC, Warfield A. Capo: Recapitulating storage for virtual desktops. In: Proc. of the 9th USENIX Conf. on File and Storage Technologies. San Jose: USENIX Association, 2011. 3–17.
- [23] Mesnier M, Chen F, Luo T, Akers JB. Differentiated storage services. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles. Cascais: ACM Press, 2011. 57–70. [doi: 10.1145/2043556.2043563]

附中文参考文献:

- [18] 杨濮源,金培权,岳丽华. 一种时间敏感的 SSD 和 HDD 高效混合存储模型. 计算机学报, 2012,35(11):2294–2305. [doi: 10.3724/SP.J.1016.2012.02294]
- [19] 尹洋,刘振军,许鲁. 一种基于磁盘介质的网络存储系统缓存. 软件学报, 2009,20(10):2752–2765. <http://www.jos.org.cn/1000-9825/3427.htm> [doi: 10.3724/SP.J.1001.2009.03427]
- [20] 刘洋. 层次混合存储系统中缓存和预取技术研究[博士学位论文]. 武汉: 华中科技大学, 2013. [doi: 10.7666/d.D608789]



唐震(1990 -),男,江苏苏州人,博士生,主要研究领域为网络分布式计算与软件工程.



魏峻(1970 -),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.



吴恒(1983 -),男,博士,副研究员,CCF 专业会员,主要研究领域为网络分布式计算,软件工程.



黄涛(1965 -),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.



王伟(1982 -),男,博士,副研究员,CCF 专业会员,主要研究领域为网络分布式计算,软件工程.