软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(2):466−480 [doi: 10.13328/j.cnki.jos.004878]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于聚类分组的虚拟机镜像去冗余方法*

徐继伟[1,2,3], 张文博[1], 魏 峻[1,2,3], 钟 华[1,3], 黄 涛[1,2,3]

[1](中国科学院 软件研究所 软件工程技术中心,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[3](中国科学院大学,北京 100190)

通讯作者: 徐继伟, E-mail: xujiwei@otcaix.iscas.ac.cn

摘 要: 随着云计算的兴起,虚拟化技术使用也越来越广泛,虚拟机正逐步取代物理机,成为应用服务的部署环境.出于灵活性、可靠性等方面的需求,虚拟机镜像急剧增长,如何高效地、经济地管理这些镜像文件已成为一个很有挑战性的研究热点.由于虚拟机镜像之间存在大量重复性的数据块,高效的去冗余方法对于虚拟机镜像管理至关重要.然而,传统的去冗余方法由于需要巨大的资源开销,会对平台中托管的虚拟机性能造成干扰,因而并不适用于云环境.提出了一种局部去冗余的方法,旨在优化镜像去冗余过程.其核心思想是:将全局去冗余变成局部去冗余,从而降低去冗余算法的空间复杂度,以达到减少操作时间的目的.该方法利用虚拟机镜像相似性作为启发式规则对虚拟机镜像进行分组,当一个新的镜像到来时,通过统计抽样的方法为镜像选取最为相似的分组进行去冗余.实验结果表明:该方法可以通过牺牲 1%左右的存储空间,缩短 50%以上的去冗余操作时间.

关键词: 云计算;虚拟化;虚拟机镜像;存储;去冗余

中图法分类号: TP316

中文引用格式: 徐继伟,张文博,魏峻,钟华,黄涛.一种基于聚类分组的虚拟机镜像去冗余方法.软件学报,2016,27(2):466−480. http://www.jos.org.cn/1000-9825/4878.htm

英文引用格式: Xu JW, Zhang WB, Wei J, Zhong H, Huang T. Virtual machine image deduplication method based on clustering. Ruan Jian Xue Bao/Journal of Software, 2016,27(2):466−480 (in Chinese). http://www.jos.org.cn/1000-9825/4878.htm

Virtual Machine Image Deduplication Method Based on Clustering

XU Ji-Wei[1,2,3], ZHANG Wen-Bo[1], WEI Jun[1,2,3], ZHONG Hua[1,3], HUANG Tao[1,2,3]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Virtualization technology is becoming more and more prevalence with the rise of cloud computing. The physical machines for service hosting are gradually being replaced by virtual ones. Driven by reliability and flexibility considerations, virtual machine images increase sharply, and how to manage them efficiently and economically has become a big challenge. Since large amount of duplicated data blocks exist in different virtual machine images, an efficient deduplication method is vital to the virtual machine image management. The existing deduplication works are not very suitable for cloud environments as they employ time-consuming algorithms which can cause

* 基金项目: 国家自然科学基金(61402450); 国家科技支撑计划(2013BAH45F01); 国家高技术研究发展计划(863)(2013AA041301); 北京市自然科学基金(4154088)

Foundation item: National Natural Science Foundation of China (61402450); National Key Technology Research and Development Program of China (2013BAH45F01); National High-Tech R&D Program of China (863) (2013AA041301); Beijing Natural Science Foundation of China (4154088)

收稿时间: 2014-04-23; 修改时间: 2014-12-31; 采用时间: 2015-08-11; jos 在线出版时间: 2015-11-15

CNKI 网络优先出版: 2015-11-16 09:22:17, http://www.cnki.net/kcms/detail/11.2560.TP.20151116.0922.001.html

serious performance interference to the neighboring virtual machines. This paper proposes a local deduplication method which can greatly optimize the deduplication process of virtual machine. The main idea of the method is to convert the global deduplication to a local one, thus considerably reducing the space and time complexity. In this method, the images are classified into different groups through an improved *k*-means clustering algorithm according to image similarities. When a new image is entered, a sampling method is used to choose an appropriate group to perform the deduplication operation. Experiments show that this approach is robust and effective. It can significantly reduce (more than 50%) the performance interference to hosting virtual machine with an acceptable increase (about 1%) in disk space usage.

Key words: cloud computing; virtualization; virtual machine image; storage; deduplication

云计算作为一种新兴的计算模式,具有弹性供给资源和按使用付费等特点.云计算通过网络提供对后台共享资源池(服务器、网络、存储、应用软件、服务等)的访问和使用,并对不同租户的资源访问进行隔离,以保证不同租户的数据安全和性能隔离.虚拟化技术[1]已经成为构建云计算环境的关键技术,国内外主流的公有云平台(如亚马逊 AWS、微软 Azure、阿里云等)都采用虚拟化技术搭建,开源社区私有云建设方案(如 OpenNebula, OpenStack,Eucalyptus 等)也都采用虚拟化技术作为支撑.虚拟化技术能够使虚拟机像物理机一样运行,虚拟机的磁盘信息(如操作系统、应用软件、用户配置、用户数据等)都封装在虚拟机镜像中.因为虚拟机镜像具有封装性,所以镜像在存储过程中以一个整体的形式存在,这就不可避免地存在相当程度的数据冗余.以亚马逊 AWS 为例,虚拟机机器镜像的持久化存储是以对象的形式保存在亚马逊简单存储服务(Amazon S3)中.隶属于不同用户的镜像可能继承自相同的模板,这使得不同镜像之间存在相同的数据块;即使镜像是各自创建而不是继承自相同的模板,镜像之间也可能存在相同的操作系统、应用软件等内容.研究发现[2],在虚拟机镜像之间存在 80%以上的数据冗余.IBM Pulse 2012 的报告指出,企业环境中存在大量镜像且不断增长.在大型企业中,虚拟机的数量高达 5 000~20 000 个,每天产生的镜像数量多达数千个,且以每两年增长 3 倍的速度增加.假设每个镜像大小为 10GB~20GB,平均每天产生的数据增量超过 10TB,这种数据存储需求严重超出了存储设备能力的增长,并且使企业管理数据的成本越来越高,数据中心的能耗越来越大.

重复数据删除技术是一种有效的消除冗余数据的技术,广泛应用于数据归档和备份中[3].传统环境中的数据去冗余技术面向的是封闭的归档或备份系统,在去冗余操作过程中具有资源独占的特性,因而更加关注数据压缩率;而云计算环境是一种开放的、资源共享的环境,虚拟机镜像去冗余操作会不可避免地对平台中托管的其他应用产生性能干扰.与此同时,我们还发现,虚拟机镜像去冗余过程中存在“相似相容”的特点,即,对拥有相同或相似操作系统、文件系统和应用软件的镜像进行去冗余之后得到的数据总量要远远小于两个镜像总大小,而对拥有不同操作系统或文件系统的镜像进行去冗余之后所得到的数据总量与两个镜像总大小基本一致.因此,在特定镜像的去冗余过程中,不相似的镜像数据将成为无效数据.我们把利用所有已有镜像数据对特定镜像进行去冗余的过程称为全局去冗余过程.在全局去冗余过程中,无效数据将占用大量的内存空间,降低有效数据查找的命中率,延长去冗余操作的时间,加重镜像去冗余操作对平台托管应用性能的干扰.

我们利用虚拟机镜像之间的相似性作为启发式规则,采用聚类分组的方法过滤镜像去冗余过程中的无效数据,解决镜像去冗余时间长、对平台中托管虚拟机干扰严重的问题.该方法可以将全局去冗余问题转化为局部去冗余问题,降低去冗余操作的空间复杂度.论文的贡献主要体现在以下几个方面:(1) 利用镜像相似性作为启发式规则对虚拟机镜像进行聚类分组,过滤去冗余过程中的干扰数据,从而降低虚拟机镜像去冗余过程中指纹查找的空间复杂度;(2) 探索了两种统计抽样方法,用于为镜像选择合适的去冗余分组,并给出了一种用于计算样本容量的可行性方法.

本文第 1 节简单介绍虚拟机镜像和分级存储的一些相关背景.第 2 节将全面阐述我们的基于聚类分组的镜像局部去冗余方法和相关的镜像抽样分组选择方法.第 3 节通过实验验证方法的有效性和稳定性.第 4 节分析介绍已有块级别去冗余工作和虚拟机镜像去冗余工作.第 5 节对本文工作进行简单的总结.

1 研究背景与动机

1.1 镜像相关概念

虚拟机镜像是封装了操作系统和应用软件的一种特殊格式的文件,可以被相应的虚拟机监控器实例化为虚拟机.虚拟机镜像格式分为两大类:全镜像模式(flat mode)和稀疏模式(sparse mode).我们常见的虚拟机镜像的格式有 RAW,QCOW,VMDK,VHD 等,其中,RAW 格式即为全镜像模式,其他格式均为稀疏模式.

虚拟机实例化是指为虚拟机镜像分配相应的计算和 I/O 资源,使其变成一台可以运行的虚拟机的过程.而虚拟机在运行过程中,会实时地从相应的镜像文件中读取和写入信息.由于存在备份、调试、错误修复等需求,系统需要周期性地(或按用户指定要求)保存虚拟机磁盘内容在特定时刻的状态,即,对虚拟机镜像文件做快照(snapshot).从文件系统语义讲,镜像文件快照是指针重定向或写时复制技术的产物,是附属于某个镜像文件存在的,但是从应用语义讲,镜像文件快照是包含虚拟机完整硬盘信息的文件.镜像快照在备份保存(跨文件系统保存)过程中会按照应用语义进行还原,这也是造成镜像冗余的重要因素.

1.2 镜像分级存储

分级存储是根据数据大小、重要程度、访问频率等指标,将数据存储在不同性能的存储设备上,并能按照特定需求实现数据客体在不同存储设备之间的自由迁移.一般情况下,存储设备的容量越大,则其存取速度与带宽越低,每 bit 价格越低.权衡容量、速度、成本三者的关系,迫使存储系统不得不从经济角度考虑分级结构.在云环境中,虚拟机镜像存储通常也会采用类似的结构.虚拟机镜像往往以对象的形式存储在容量较大、价格较低的存储服务中,而且存储服务往往具有更高的可靠性.虚拟机实例化时,需要将虚拟机镜像从存储服务中加载到本地存储设备中.当虚拟机镜像需要备份或持久化保存时,需要将虚拟机镜像保存到存储服务中.本文的去冗余工作主要是针对这种场景.

为了便于理解,本文中将复杂的存储服务抽象成简单的存储设备(如磁盘阵列),在本文后面的论述和实验部分均以存储设备替代存储服务.该方法适用的部署架构如图 1 所示,终端用户通过网络访问虚拟机提供的服务,镜像文件存储在本地存储介质中,虚拟机通过 NFS/CIFS/iSISC 等协议进行镜像文件读写.图中的备份存储表示远程存储服务,虚拟机镜像需要通过网络持久化到备份存储中.这种架构体现了当前主流的云环境搭建场景,如 AWS,CloudStack 等均采用类型架构,在 AWS 中镜像存储类似于 EBS,备份存储类似于 S3,而在 CloudStack 中镜像存储则类似于主存储,备份存储类似于二级存储.

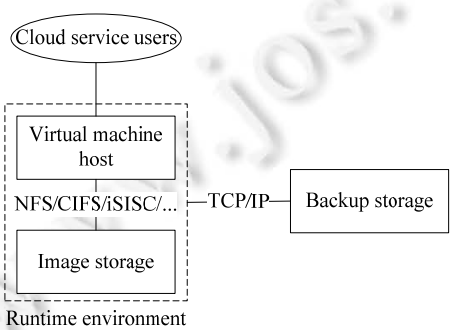


Fig.1 Typical image backup system deployment architecture

图 1 典型的镜像文件备份系统部署架构

1.3 虚拟机镜像去冗余对平台托管Web应用性能影响

为了说明研究虚拟机镜像去冗余方法的必要性,我们按照图 1 的架构搭建了一个基于虚拟化的计算平台.在平台中的一台虚拟机上运行一个 RUBiS 应用,并通过监控软件监测其性能.在应用运行过程中,我们采用传统方法对平台中的虚拟机镜像进行去冗余备份,方法步骤如下:(1) 对镜像进行分块并采用 MD5 算法计算每个分

块的哈希指纹;(2) 查找指纹库确定指纹是否已经存在,判断该数据块是否重复;(3) 存储数据块.我们对比去冗余过程执行之前和执行时的 Web 应用性能,其结果如图 2 所示.我们观察了 RUBiS 应用的两个性能指标——每秒点击数和响应时间.从图中我们可以看出,虚拟机镜像去冗余会使应用每秒点击量下降,并使应用响应时间提高,整体性能损失在 15%以上.这说明虚拟机镜像去冗余操作会对平台托管的虚拟机应用造成干扰.因此,本文关注一种可以降低性能干扰的虚拟机镜像去冗余方法.

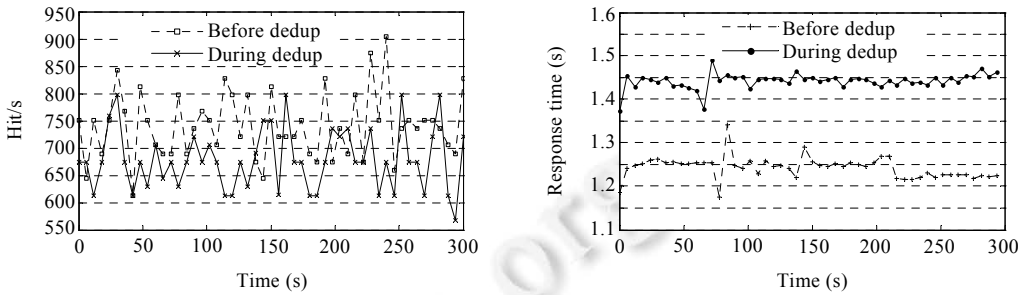


Fig.2 Comparison of Web application performance before and during backup operations

图 2 备份操作执行之前和执行时的 Web 应用性能对比

2 基于聚类分组的镜像局部去冗余方法

本文工作的目标是消除虚拟机镜像中的冗余数据.镜像去冗余存在"相似相容"的特点,拥有相同操作系统、应用和数据集的镜像有很大几率(高于 80%)拥有相同数据块;反之,操作系统不同的镜像拥有相同数据块的几率则很小(可能低于 1%).这就意味着,如果两个镜像不相似,它们之间存在相似数据块的概率就很小,对它们进行去冗余只能取得非常微小的存储收益.这是本文方法的可以有效工作的基本前提.在这样一种启发式规则下,我们通过过滤去冗余过程中的无效数据来降低查找算法的空间复杂度,以牺牲少量的存储空间为代价换取快速的去冗余时间.

假设所有已存在的虚拟机镜像形成集合Ω,我们的方法首先利用一种聚类算法 F 将Ω中的镜像分成若干小集合,$F(\Omega)=\{\Omega_1,\Omega_2,\ldots,\Omega_n\}$,其中,$\bigcup_{i=1}^{n}\Omega_i=\Omega$,$\bigcap_{i=1}^{n}\Omega_i=\varnothing$.针对每一个分组$\Omega_i$ 中的镜像,采用固定长度分块去冗余的方法执行去冗余操作,即,将Ω域的全局问题转化为Ω_i 子域的局部问题.对于新到来的镜像α,采用特定的取样方法 S 提取α的指纹样本,计算样本 $S(\alpha)$与每个分组Ω_i 的相似度 $Sim(S(\alpha),\Omega_i)$,选择相似度值最大的分组Ω_*对α执行去冗余操作,其中,$\Omega_*=\{\Omega_i=\max_{i=1}^{n}Sim(S(\alpha),\Omega_i)\}$.分组 n 个数的大小需满足条件 $n>p/m$,其中,p 表示Ω的全部指纹数据大小,m 表示可用内存大小.图 3 为局部去冗余方法示意图.接下来我们将具体介绍方法细节.

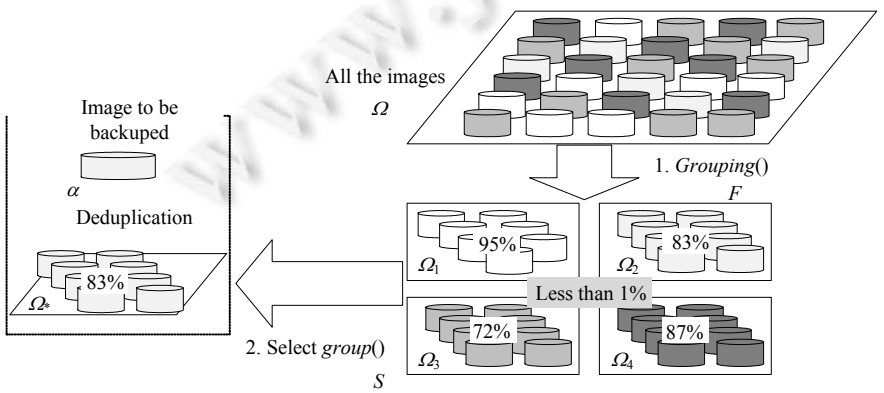


Fig.3 Local deduplication diagram

图 3 局部去冗余示意图

2.1 镜像备份及去冗余时机

去冗余操作将分别消耗大量的服务器端资源和存储端资源.考虑到备份和去冗这两个操作的时机,将会有3 种不同的策略:备份前去冗余、备份后去冗余和备份中去冗余.若执行备份前去冗余,网络传输的数据量将会是压缩后的数据大小,备份所需要的时间窗口将会缩短,然而指纹计算和指纹查找的过程就只能在主机端完成,这将会对主机上托管的虚拟机或应用性能造成严重干扰;相反地,若执行备份后去冗余,则数据传输量的大小将与镜像大小一致,备份所需要的时间窗口将会大为增加.我们的目标是平衡主机端和存储端的资源开销,避免造成负载集中的情况出现,同时要尽量降低备份所需时间窗口.因此在我们的工作中,我们采用备份中去冗余的策略.这样,数据传输量会尽可能地小,而指纹计算和指纹查找也将分别在主机端和存储端执行.

在我们的备份去冗余系统中,我们把主机端作为备份的客户端,把存储端作为备份的服务器端.如图 4 所示,我们把数据切分的模块和指纹计算模块放到主机端,而把指纹查找和数据块存储模块放到存储设备端.这样做的目的是为了减小网络传输的数据量,只需要从主机端到服务器端传输未曾保存的数据块.然而,数据切分和指纹计算是CPU 密集型操作,由于在云环境中主机端托管了大量的虚拟机,这就不可避免地会造成资源竞争,从而对托管虚拟机的性能产生影响.对此,我们能做到的就是加快去冗余备份的速度,以缩短影响时间.

因此,与之前的工作不同的是,我们在去冗余系统中加入了一个预处理模块(图 4 中灰色框),用于加速指纹查找过程,从而加快去冗余的速度.在这个模块中,我们采用简单的聚类分组算法将镜像分为若干组,使得指纹查找由一个空间复杂度高的全局问题变成一个空间复杂度相对较小的局部问题.同时,为了权衡压缩比例和指纹查找性能,我们采用分组优化方法来控制分组大小.

对于数据块的存储,我们把固定数目的数据块以追加的方式存储到存储设备端的文件中,由于数据块大小是固定的,因此只需知道数据块的编号就可以计算出该数据块起始位置在文件中的偏移.指纹到数据块的映射由指纹索引表来维护.

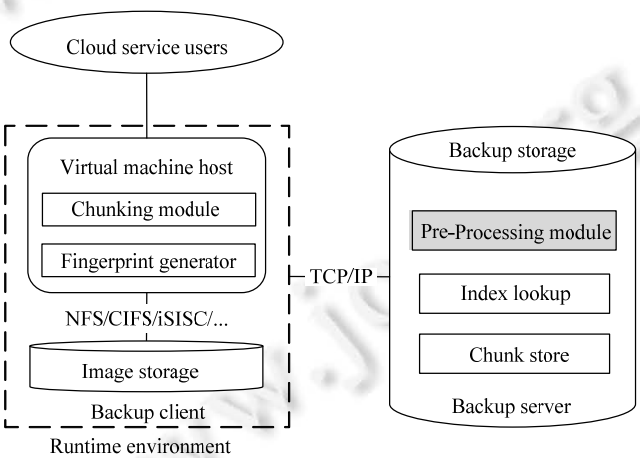


Fig.4 System deployment architecture

图 4 系统部署架构

2.2 镜像聚类分组算法

图 5 展示了镜像备份、指纹和数据块之间存在的对应关系,镜像的备份是一个逻辑的实体,可以由元数据文件和相应的数据块复合而成.元数据文件包含一组顺序的指纹,可以看做是一组指纹的集合,指纹则可对应相应的数据块.

数据块指纹集合是一维的数据集合,我们采用 *k*-均值聚类算法对指纹数据集合进行处理.我们根据镜像的特点和组织结构对 *k*-均值聚类算法[4]进行改进,并将改进的算法用于镜像元数据文件的分组操作.运行 *k*-均值

聚类算法之前,首先要确定 k 的大小.我们根据 $k>p/m$ 的原则来决定 k 的大小,并通过调整 k 的大小保证每组镜像的指纹能全部加载到内存中.

k 聚类算法的基本原理如下:给定一组元素的集合$(x_1,x_2,\ldots,x_n)$,其中,每个元素是一个 d 维的数组.k 聚类算法的目标就是将这 n 个元素划分为 k 个子集合 $S=(S_1,S_2,\ldots,S_k)$,并使所有集合中元素到集合中心点距离的平方和最小.

$$\arg\min_{S}\sum_{i=1}^{k}\sum_{x_j\in S_i}\|x_j-\mu_i\|^2 \tag{1}$$

其中,μ_i 表示集合 S_i 的中心点.

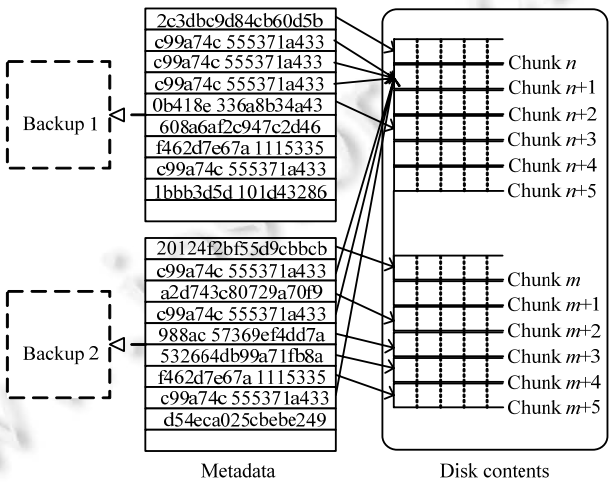


Fig.5 Relationship among backup, fingerprint and chunks

图 5 备份、指纹、数据块之间的关联关系

虚拟机镜像包含虚拟机的硬盘信息,因此,我们可以大致根据镜像的启动扇区信息判断镜像的操作系统和文件系统类型,我们把这些信息作为 k-均值聚类算法的一种先验知识.在 k-均值聚类中,初始中心点的选取对聚类的结果具有一定的影响.我们在选择初始中心点时,根据前面提到的先验知识,尽量选择具有不同操作系统和文件系统的镜像作为中心点.

在改进的聚类算法中,我们采用镜像相似度来表示两个镜像之间的距离.由于镜像内部存在大量冗余的数据块,于是在镜像的元数据文件中也会有大量相同的指纹,如 0 字节填充的数据块指纹.我们把这些冗余数据块称为内部冗余(inner duplication).这些内部冗余数据块可能因为数量过多而影响两个镜像之间相似度的计算.要计算镜像相似度,首先要消除镜像内部冗余数据块,这个过程被称为内部去冗余(inner deduplition);与之相对的是消除镜像之间的冗余数据块,称为外部去冗余(inter deduplication).

因为元数据文件包含了整个镜像的所有数据块指纹,所以要计算两个镜像的相似度可以等价为计算其相应的元数据文件的相似度.正如前文中提到那样,一个元数据文件可以被视为一组指纹向量,即 $M=\langle f_1,f_2,\ldots,f_n\rangle$.我们用 $M'=(f_1,f_2,\ldots,f_m)(m\leqslant n)$来表示元数据文件去冗余后的指纹集合.我们把 M'称为镜像的特征集合.于是,我们可以通过公式(2)计算两个镜像 A,B 之间的相似度.

$$Sim(A,B)=\frac{2\times|M'_A\cap M'_B|}{|M'_A|+|M'_B|} \tag{2}$$

根据公式(2)可知,两个镜像之间的相似度取值范围为[0,1].

于是,我们给出虚拟机镜像聚类算法.

算法 1. 虚拟机镜像聚类算法.

输入:初始分组数 k;c_i 代表 i 个分组的选出初始中心点.

算法:

1: Set $S_1=\{c_1\},S_2=\{c_2\},\ldots,S_k=\{c_k\}$
2: repeat
3: $S_1' \leftarrow S_1, S_2' \leftarrow S_2, ..., S_k' \leftarrow S_k$
4: for all $A \in \Omega$ do
5: *minindex*, *minvalue*←1, *loop*←0
6: for each $j \in [1,k]$ do
7: $B \leftarrow c_j$
8: *value*←*Sim*(*A*,*B*) //计算镜像 *A*,*B* 的相似度
9: if *value*<*minLengh* Then
10: *minvalue*←*value*
11: *minindex*←*j*
12: end if
13: end for
14: $S_{minindex} \leftarrow S_{minindex} \cup \{A\}$
15: end for
16: for each $i \in [1,k]$ do
17: $c_i \leftarrow RepickCentroid(S_i)$ //重新选择分组 S_i 的中心点
18: end for
19: *loop*←*loop*+1
20: until ($S_1' == S_1$ and $S_2' == S_2$ and ... and $S_k' == S_k$) or (*loop*<*t*)

众所周知,*k*-均值聚类算法的时间复杂度为 $O(tknm)$,其中,t 为迭代次数,k 为分组的数目,n 为镜像总数,m 为数据维度.而在我们的算法中,数据维度为 1,于是,算法的时间复杂度为 $O(tkn)$.而在一般情况下,$k<<n$ 并且 $t<<n$,即,算法可在多项式时间内完成.

为了避免不同的分组数量对局部去冗余的效果以及内存的使用的影响,我们从触发时机和分组合并两个方面对算法进行优化.

1) 触发时机

由于聚类算法需要的时间复杂度较高,如果运行过于频繁则会导致大量的资源浪费,影响系统性能;反之,如果运行次数过少则可能导致其失去原有的作用,造成镜像去冗余备份存储中的内存溢出.因此,我们需要设定合适的触发时机.假设镜像分为 k 组,可用内存大小为 M_s,对于每一个分组 $i(0<i\leqslant k)$,如果分组 i 总的特征值大小超过可用内存 M_s,则对 i 分组使用分组聚类算法.这样既可以做到在去冗余过程中不会造成内存溢出,又可以做到聚类算法的时间复杂度始终控制在一个合理的范围内.

2) 分组优化

通过实验我们发现,尽管算法可以成功地将镜像分为若干组,然而每组镜像的特征值总量大小相差较大,有的分组只有少数几个镜像存在.为了不致影响取样比较阶段的性能以及提高去冗余率,我们需要在不超过内存限制的前提下,将一些小的分组进行合并.

从某种角度来讲,基于镜像相似的聚类可能存在逻辑上的冲突,例如,镜像 A 和镜像 B 的相似度大于 50%,镜像 B 和镜像 C 的相似度大于 50%,然而镜像 A 和镜像 C 的相似度可能为 0.然而,我们聚类的目标并不仅仅是为了使分组中所有的镜像具有极高的相似度,也尽可能地将每个分组的指纹大小控制在可使用的内存范围之内.在这种目标前提下,不同的分组之间也可能存在相似度极高的镜像.同样,由于存在分组优化,同一个分组中

的镜像也可能完全不具有相似性,而这也是为了最大限度地利用内存资源,加速去冗余过程.

2.3 取样比较选择去冗余分组

完全的内存去冗余是本文方法最主要的特征,我们已经通过聚类算法实现镜像的分组.当一个新的镜像到来时,系统将根据特定的规则 R 选取一组指纹样本.然后计算该组样本在每一个去冗余分组中的命中率,并选择命中率最高的分组来执行去冗余操作.在我们的工作中,我们对比了简单随机取样和系统取样两种方法.

1) 简单随机取样(SRS):首先,按照固定大小(4KB)将镜像分为 N 份(即样本空间为 N),并分别用[1,N]为每一份进行编号;然后,随机地从[1,N]中选择 n 个数字(即样本容量为 n),计算对应的这 n 个块的指纹,形成一个样本 S.

2) 系统取样(SS):和简单随机取样一样,首先将镜像分为大小相等的 N 份;然后,将从 1 到 N 按顺序等分成 m 组,每组有 N/m 个块,分别用[1,N/m]进行编号;从每组中随机选择 n/m 个块,计算这些块对应的指纹,形成一个样本 S.

为了提高样本在每个分组中命中率的辨识度,我们需要计算样本 S 的容量 n.假设样本在分组的命中率为 p,则其未命中的概率为 $1-p$.令变量 X 代表样本在分组中的命中数,则 X 服从参数为(n,p)的二项分布,记作 $X\sim B(n,p)$.当 n 足够大时,二项分布 $B(n,p)$可以被近似为均值为μ、方差为σ^2 的正态分布,记为 $N(\mu,\sigma^2)$,其中,

$$\mu = np, \sigma = \sqrt{np(1-np)}.$$

于是,令 $y=(x-\mu)/\sigma$,则 Y 服从标准正态分布,记为 $Y\sim N(0,1)$,其概率密度函数为

$$\Phi(y) = \int_{-\infty}^{y} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{y^2}{2}} \mathrm{d}y \tag{3}$$

假设样本在分组中至少命中 x 个的概率为ρ,即 $\mathrm{P}(X\geqslant x)=\rho$,那么,

$$\Phi(y)=\mathrm{P}(X<x)=1-\rho \tag{4}$$

于是,可以得到 $x=\sigma\cdot\Phi^{-1}(1-\rho)+\mu$.综合公式(3)、公式(4),可以得到有效样本的计算公式(5).

$$n' = \frac{2xp - [\Phi^{-1}(1-\rho)]^2 pq + \sqrt{[[\Phi^{-1}(1-\rho)]^2 pq - 2xp]}}{2p^2} \tag{5}$$

由于镜像中存在冗余数据,因此在实际取样过程中可能存在冗余样本.我们假设镜像的内部冗余率为 r,那么实际样本容量可以根据公式(6)计算.

$$n = \frac{2xp - [\Phi^{-1}(1-\rho)]^2 pq + \sqrt{[[\Phi^{-1}(1-\rho)]^2 pq - 2xp]}}{2p^2 r} \tag{6}$$

上述公式用于计算取样样本的大小,用于给出一种可供参考的选择.由于事先并不知道哪一个分组是最合适的分组,在参数选择的过程中,需要根据历史经验进行判断.如要求样本有 99%的概率在分组中至少命中 100 条,假定分组数据冗余率为 40%,样本命中的概率为 30%,则ρ=0.99,p=0.4,x=100,r=0.3.

通过统计取样的方法,我们可以为新来的镜像选择合适的去冗余分组执行去冗余操作,从而过滤掉干扰分组,提高查找命中率.

2.4 方法的时间收益与空间开销

基于聚类分组的镜像局部去冗余方法根据镜像相似度将镜像分为若干组,使得每组镜像的指纹可以被完全地载入内存中,从而避免了传统去冗余方法频繁的内存/磁盘数据交换导致的性能瓶颈,可以大大加速去冗余过程.该方法取得的收益是时间维度的收益.然而,由于我们只进行分组内去冗余,这必然会导致一定量的冗余数据块的产生,这意味着方法造成的开销是空间维度的开销.在时间维度上,去冗余时间随着数据量的不断增大而不断增长变为常数时间.接下来,我们将分析在空间维度上方法造成额外开销的上界.

就全局而言,我们假设共有 n 个不同的数据块,每个数据块的冗余份数为 c,则去冗余之前的数据块为 $\sum_{i=1}^{n} c_i$, 即 $\overline{c}\times n$, 则全局去冗余率可表示为

$$r=1-\frac{1}{\bar{c}}.$$

我们将所有数据块分为 m 组,并假设每个数据块平均在 $\bar{m}(1<\bar{m}<m)$ 个分组中出现,那么去冗余之后的数据块总数为

$$r=1-\frac{\bar{m}}{\bar{c}}.$$

即,$\frac{\bar{m}}{\bar{c}}$ 的比值越小,方法造成的额外开销就越小.

3 实验验证

我们通过一系列的实验来验证我们的方法的有效性和性能.我们从 OnceCloud[5]云环境中选择了 584 个不同的虚拟机镜像,每个大小在 15GB~20GB 之间.其中包括 raw 格式镜像 416 个,vhd 格式镜像 168 个,镜像总大小为 6.68TB,在实验中可用于去冗余的内存大小为 500MB.我们采用 128 位的 MD5 值作为数据块指纹,使用 64 位地址作为块索引,那么一个块记录需要 192 位(24B).如果块大小为 4KB,则需要 40.2GB 的空间来存储所有的指纹数据.即便是所有的冗余数据块被去除之后,指纹数据总量仍大于可用内存大小,因此无法全部放置到内存中.为了与已有工作做对比,我们首先实现了一个基于硬盘的哈希表,用来执行全局去冗余操作(global dedup),在去冗余过程中,需要不断地进行硬盘读写,实现内存数据交换.然后,我们用基于镜像聚类分组的方法执行局部去冗余操作,并对两种方法的实验结果进行比较.实验中,采用 5 台刀片服务器作为实验设备,每台服务器拥有两颗 Intel Xeno E5645 CPU,600GB 硬盘和 32GB 内存.备份存储端采用 10TB 空间的存储集群.所有设备通过千兆网络设备进行连接.

3.1 去冗余率和操作时间

在虚拟机镜像去冗余中,去冗余率和操作时间是我们最为关心的两个要素,前者关系到数据最终占用的存储空间大小,后者则直接影响到去冗余备份时间窗口大小.在本节中,我们将对这两个要素进行对比.为了验证本文方法的先进性,我们先不对镜像数据进行分组,直接采用基于硬盘哈希表的方式进行全局去冗余,然后再采用基于镜像聚类分组的方法进行局部去冗余.去冗余率的实验结果如图 6 所示,我们把所有镜像数据集的总大小(before dedup)记为 100%,当内部去冗余(inner dedup)完成后,数据集大小变为 24.2%.从图中我们可以看出:全局去冗余方案(global dedup)可以达到 9.2%的数据压缩率,而基于 k-均值聚类分组的局部去冗余方法(local dedup)可以达到 10.2%的数据压缩率,与全局去冗余仅有 1%的差距.相对于压缩掉的 90%的数据,1%的差距完全在可接受范围之内.在接下来的实验数据中我们可以看到,1%的空间消耗将会节省成倍的时间开销.

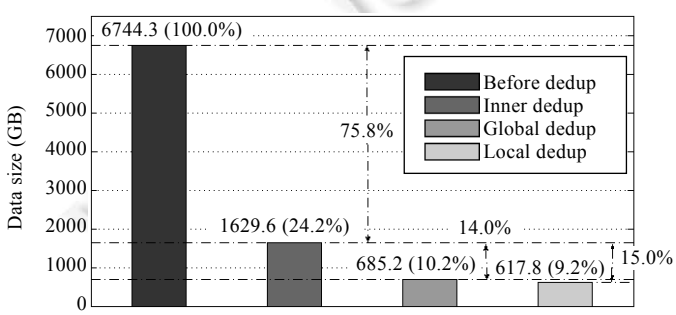


Fig.6 Comparison of data size and deduplication rate under different approach

图 6 不同方案下数据大小及数据压缩率对比

在完成 584 个镜像的去冗余之后,我们分别对两种不同格式(raw 和 vhd)的新镜像进行去冗余操作.对于每种镜像,我们分别采用全局去冗余方法和基于 k-均值聚类分组的局部去冗余方法备份 20 次,备份时间如图 7 所

示.对于 raw 格式和 vhd 格式镜像,我们的局部去冗余方法都将节省超过 50%的时间.由于我们的方法是一种完全的内存索引查找,在去冗余过程中无需与外存进行进行交换,与全局去冗余方法相比,可以节省大量磁盘查找的时间.

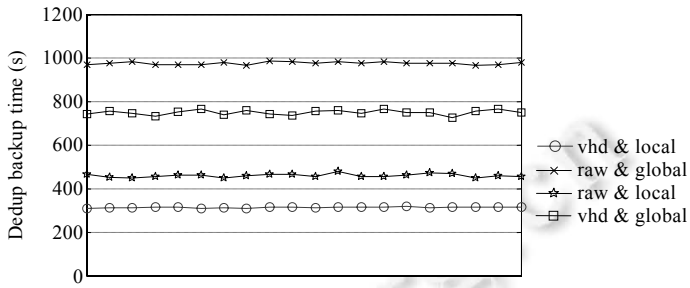


Fig.7 Different format image deduplication time

图 7 不同格式镜像去冗余时间

从上面的实验结果可以看出,我们的基于 *k*-均值聚类分组的局部去冗余方法可以在以付出微小的空间开销(1%左右)为代价的基础上大幅度降低时间开销(超过 50%).

3.2 聚类分组算法的有效性和稳定性

由于聚类算法的结果具有一定的不确定性,这就意味着我们采用聚类分组的方法去冗余可能导致最终结果具有一定的随机性.为了保证算法的可用性,我们需要对算法的有效性和稳定性进行检验.在这部分的实验中,我们使用聚类分组算法将镜像分为 *k* 组,并进行去冗余操作.重复运行该操作 10 次,对于每一次实验,我们计算数据压缩率和每个分组的统计学特性.图 8 展示了每次实验的数据压缩率,结果显示,最终数据压缩率稳定在 89.5%~89.9%之间.这表明我们的基于 *k*-均值聚类分组的局部去冗余算法在实验给定的数据集上具有很好的稳定性.实验的平均压缩率为 89.7%,仅比全局去冗余低 1.1%,所有实验去冗余结果与全局去冗余相比低不超过 1.5 个百分点,这表明我们的算法具有很好的有效性.

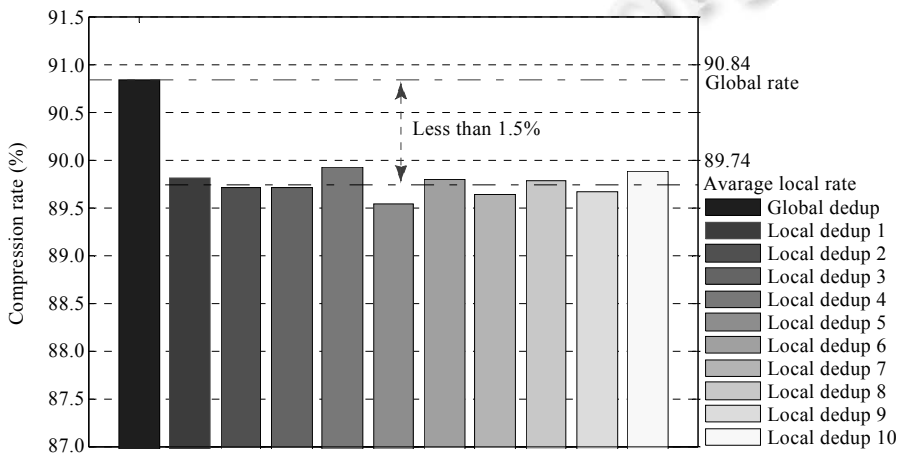


Fig.8 Data compression rate comparison of ten tests

图 8 10 次实验的数据压缩率对比

根据给定的数据集大小和实验内存限制,我们的算法将所有镜像分为 7 组,接下来我们来看这几组镜像的统计学特点.我们统计了每个分组中的中心点镜像与其他镜像的相似度,选择其中的最大值、最小值、均值、中位数和标准差.最大值和最小值反映出该分组中镜像与中心点镜像的最大和最小相似度;标准差反映出相似

度分布的偏离状况;均值反映出分组中所有镜像的平均相似程度;中位数可以用于反映分组内镜像相似度的偏斜程度.图 9 从 10 次实验结果中随机选择了 6 次进行展示,括号中的数字 n 代表第 n 个结果.纵轴表示的是数据值,而横轴 x 代表第 x 个分组.下面我们分析一下这些统计特征值的实际意义.图 9(a)中的第 1、第 2 和第 4 个分组拥有很高的相似度,因为其最大值、最小值和中位数都非常高,而方差却非常低.图 9(b)中的第 3 个分组最小值非常小(低于 5%),并且方差也比其他分组要高,但我们仍然认为该分组的大部分镜像具有很高的相似度,其原因是该分组的最大值、中位数和均值都处于较高水平.与之相对的第 5 个~第 7 个分组则拥有相对较低的相似度,即便如此,其相似度仍然超过 10%,有的甚至超过 20%,这也说明我们方法具有很好的有效性.

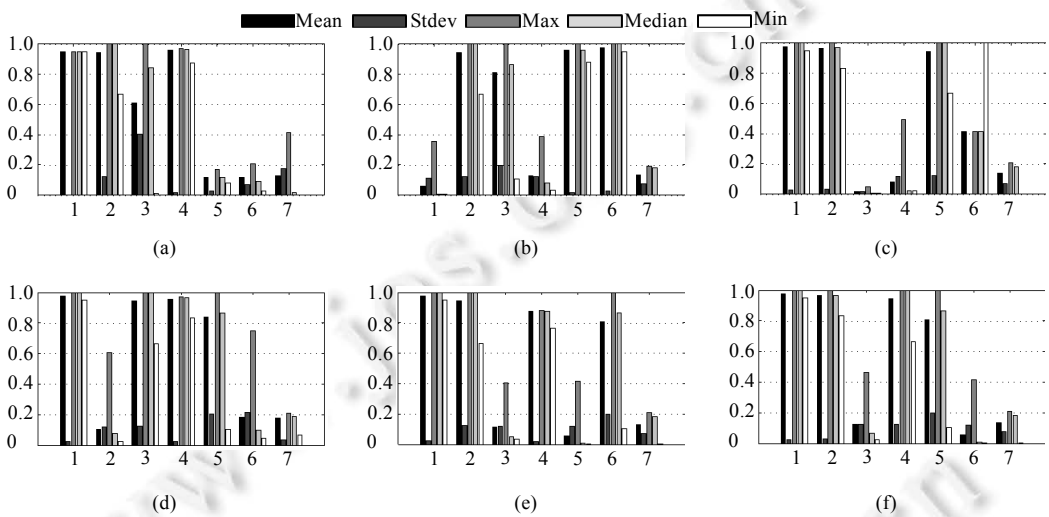


Fig.9 Each group's similarity statistical characteristic

图 9 分组相似度的统计学特征

通过这一节的实验结果,我们证明了基于 k-均值聚类分组的局部去冗余方法就有很好的有效性和稳定性,在实践中具有指导性意义.

3.3 抽样方法检验

前面已经提到:我们对需要去冗余存储的镜像分别采用简单随机抽样和系统抽样方法进行抽样,从而根据抽样在各个分组中的命中率选择合适的去冗余分组.我们分别对 vhd 格式镜像和 raw 格式镜像进行去冗余实验.对于每种镜像,我们进行 10 次抽样并计算抽样在各个分组中的命中率,以此来验证抽样方法的有效性.对于 vhd 格式镜像和 raw 格式镜像,我们分别设置参数:

- ρ=0.99,p=0.4,x=100,r=0.4;
- ρ=0.99,p=0.4,x=100,r=0.112.

根据公式(6),可计算出样本容量分别应该为 768 和 2 672.图 10 分别给出了两种镜像在两种方法下的取样命中率和实际的命中率.由图中所示数据我们可以看出,通过两种抽样方法计算出的命中率与实际命中率都十分接近.根据图 10 中 vhd 格式镜像在各个分组中的命中率数据,我们可以推断出应该用第 4 个分组对该镜像进行去冗余.同样地,对于 raw 格式镜像,应该采用第 5 个分组进行去冗余.

从上面的实验中,我们可以发现一些有趣的现象:

- 首先可以看到:镜像除了在某个分组中拥有最高命中率外,在其他分组中也可能会有较高的命中率.以 vhd 格式为例,镜像在分组 4 中具有最高的命中率(约为 75%),同时,在分组 7 中也有较高的命中率(约为 38%).由于镜像只在分组 4 中进行去冗余,于是会导致分组 4 和分组 7 之间存在冗余数据块,这也是局部去冗余的劣势所在.

- 其次,通常情况下会认为,系统抽样相对于简单随机抽样,其结果会更加精确,但是在我们的实验中,可以认为两者在选择镜像分组中具有相同的效果.从某种角度来讲,这与我们选取了足够的样本容量有一定的关系.

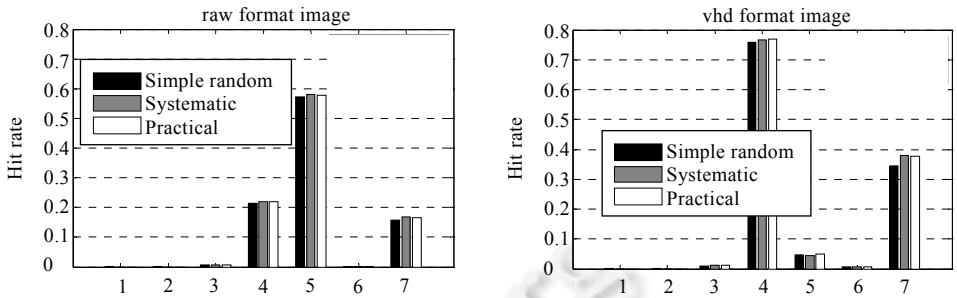


Fig.10 Sample hit rate vs. practical hit rate

图 10 样本命中率和实际命中率

我们的工作还对样本容量对取样准确性的影响进行了研究.我们用前面的 raw 格式镜像进行实验,样本容量从 100 开始,并以每组实验增加 50 的幅度递增.每组实验进行 100 次取样,计算样本误差率,然后得到平均误差和最大误差,实验结果如图 11 所示.图中横轴代表样本容量,纵轴代表抽样误差.从图中我们可以看出,样本大小与取样命中率的精确度有密切的关系.样本容量越大,则取样计算的结果越发精确.图 11 中,虚线所示为我们根据公式(6)计算所得数值,而在此之前,样本最大误差率已经达到一个相对稳定的状态.这就意味着,我们计算所得的数值可能会略大于实际需求.然而,我们的目的是为了提供一个参考值,在实际测量之前是无法去精确预测所需要的样本大小的.

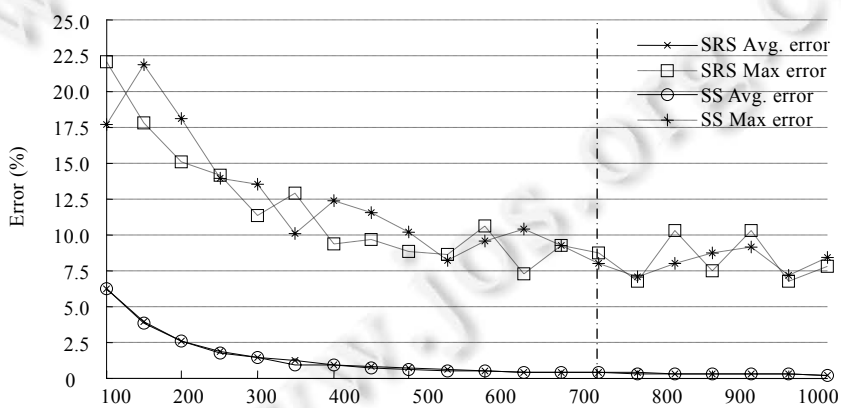


Fig.11 Relationship between sample error and sample size

图 11 样本误差与样本容量关系

3.4 实验小结

我们共选取 vhd 和 raw 两种不同格式镜像 584 个进行实验,通过实验对比传统去冗余方法和基于聚类分组的镜像去冗余方法的去冗余率和去冗余时间,证明我们的方法存在更加适合云计算的场景.我们利用基于聚类分组的镜像去冗余方法进行多次实验,其结果误差控制在 0.4%以内,说明我们的方法具有很好的稳定性和有效性.通过抽样命中率结果和真实命中率结果,也证明了我们给出的计算样本容量的方法具有一定的参考价值.

4 相关工作

去冗余技术本质上是一种数据压缩技术,可以用于消除单个或多个数据文件中的重复冗余数据,从而保证相同的数据只存在 1 份,以达到降低存储空间、提高存储利用率的目的.根据具体的实现级别,可以将去冗余方法分为两类:一类是文件级别去冗余[6,7],另一类是块级别去冗余[8−15].

4.1 块级别去冗余

块级别去冗余大致可分为 3 部分的操作:

(1) 数据分块与指纹计算,将数据文件切分成独立的数据块(chunk),并通过哈希算法计算这些数据块的指纹信息,利用指纹为这些数据块建立索引,并形成数据指纹库.

(2) 指纹查找,根据数据块指纹是否已经存在来判断数据块是否冗余.

(3) 数据块存储,通过合理的方式将数据块保存到存储设备中.

数据分块相关工作主要分为两类:一类是固定长度切分,另一类是可变长度切分.例如,Rsync[16]和 Venti[8]都是基于固定长度切分,将原始数据切成等大小的数据块.由于固定长度切分可能导致边界漂移问题[17],影响去冗余效果,因此也有一部分工作致力于研究快速的变长切分方法,如,Rabin 指纹算法可以在多项式时间内计算变长数据指纹[18].在数据块指纹计算时,为避免哈希碰撞,大部分工作都采取 MD5[19]或 SHA-1[20]算法.由于 MD5 或 SHA-1 计算复杂度高,需要消耗大量的 CPU 资源,而其计算得到的指纹占用较大空间,使得在数据块冗余判断时容易导致内存瓶颈.Rsync[16]则采用轻量级与重量级哈希算法相结合的方式,以降低计算开销.Policroniades 等人[21]对文件哈希、固定长度哈希和可变长度哈希算法进行了比较,实验结果显示,三者对相同大小数据进行处理的时间分别为 62s,71s 和 340s.即,可变长度切分去冗余将大幅度提高操作的时间复杂度.

指纹查找是去冗余操作的关键步骤,需要从海量指纹数据中判断某一特定的指纹是否存在,并持续对指纹库进行更新.由于指纹数据量巨大,内存无法容纳,因此需要将数据放置在硬盘中.而由于硬盘读写速度瓶颈的限制,指纹查找操作导致频繁的硬盘读写将极大地影响指纹查找的性能,从而降低去冗余操作的效率.在 Venti 系统中,这个问题导致至少 50%的性能下降.Zhu 等人[12]将这个问题称为硬盘瓶颈问题.为了解决这个问题,他们采用 Bloom Filter 算法[22]来判断数据块是否冗余.Zhu 等人认为,使用这种方法,1GB 内存可以支持 1 000 000 000 个数据块的查询,若按数据块大小为 4KB 计算,则可支持 4TB 大小的去冗余后数据.与此同时,该方法的假阳性概率约为 2.17%~2.40%,在镜像存储中,如此高的错误概率是无法让人接受的,其导致的直接结果就是镜像数据丢失,甚至是镜像不可用.而采用 MD5 算法计算数据指纹,其哈希冲突的概率只有 10^{-9}[17];采用 SHA-1 算法,其哈希冲突的概率则更低.

Lillibridge 等人[11]认为,块具有局部性(locality)特征.他们把数据文件分成相对较大的段(segment),对于每一个段,选取与之前最相似的段进行去冗余.通过取样的方式,从每段中选取若干块组成该段样本.寻找相似段时,将所有段的样本加载到内存,通过对比选择最相似的样本.之后,将该样本对应的段指纹加载到内存中进行去冗余.Lillibridge 等人认为,使用这种方法,只需两次磁盘读写操作就可以完成冗余数据块的判断,从而缓解磁盘瓶颈问题.数据指纹是对数据块的摘要,Lillibridge 等人的方法的本质是对数据指纹的摘要,虽然这从一定程度上缓解了磁盘瓶颈问题,却无法从根本上解决该问题.在云环境中,镜像数据是呈指数级增长的,这就意味着指纹数据也将呈指数级增长,在取样概率不变的情况下,指纹数据的样本也会呈指数级增长.而内存增长却遵循摩尔定律呈线性增长,在这种情况下,指纹样本也可能会面临指纹一样的困境,即,无法一次性载入内存,需要进行多次磁盘读写.

4.2 虚拟机镜像去冗余

随着虚拟化的广泛应用,出现了大量针对虚拟机镜像的去冗余工作.Jin 等人[2]将去冗余技术引入到虚拟机镜像存储中,并通过实验证明,在针对虚拟机镜像去冗余的过程中,固定长度切分和可变长度切分具有几乎相同的数据压缩率.Clements 等人[23]提出了针对存储区域网络(SAN)的镜像去冗余存储方案 DeDE.该方案的特点是去中心化去冗余存储.

Zhang 等人[24]致力于权衡去冗余率和去冗余效率,通过降低去冗余率来提高去冗余效率.然而,Zhang 等人的工作只是针对虚拟机镜像快照的备份,而我们的方法则可以适用于所有形式的镜像.我们的工作主要是在去冗余过程之前提供一种预处理方法,而去冗余过程则是采用一种常规的方法[8].因此,文献[24]与我们的研究在关键过程上并无冲突,可以同时工作,以提高去冗余效率.

5 总 结

在云环境中,将去冗余技术用于虚拟机备份存储中,可以减少虚拟机镜像备份过程中对于存储空间需求.然而,去冗余的过程可能导致平台中托管虚拟机的性能下降.在我们的实验中,镜像去冗余操作可能导致 Web 应用性能下降 15%~20%.为了降低这种性能影响,我们利用虚拟机镜像去冗余过程中"相似相容"的特点,对传统的去冗余技术进行改进,创造性地加入了去冗余的预处理模块.利用 k-均值算法对虚拟机镜像进行聚类分组,将全局去冗余变为局部去冗余,降低了指纹查找的空间复杂度,从而实现完全的内存去冗余,使去冗余操作的时间大幅度降低.我们利用抽样的方法对要备份的镜像进行取样,计算样本在各个分组中的命中率,从而实现去冗余分组的选择.实验结果显示,我们的方法稳定、有效,能够以牺牲 1%左右存储空间为代价,降低 50%以上的时间开销.在今后的工作中,我们将通过实验验证本文方法及其他去冗余加速方法的兼容性及效果.

References:

[1] Goldberg RP. Survey of virtual machine research. Computer, 1974,7(6):34−45. [doi: 10.1109/MC.1974.6323581]

[2] Jin K, Miller EL. The effectiveness of deduplication on virtual machine disk images. In: Proc. of the Israeli Experimental Systems Conf. ACM Press, 2009. 66−72. [doi: 10.1145/1534530.1534540]

[3] Ao L, Shu JW, Li MQ. Data deduplication techniques. Ruan Jian Xue Bao/Journal of Software, 2010,21(5):916−929 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3761.htm [doi: 10.3724/SP.J.1001.2010.03761]

[4] Hartigan JA, Wong MA. Algorithm AS 136: A *K*-means clustering algorithm. Journal of the Royal Statistical Society, Series C (Applied Statistics), 1979,28(1):100−108. [doi: 10.2307/2346830]

[5] Institute of Software Chinese Academic of Sciences. OnceCloud: A cloud service platform. 2014. http://www.once.com.cn/OncePortal/oncecloud

[6] Bolosky WJ, Corbin S, Goebel D, Douceur JR. Single instance storage in Windows 2000. In: Proc. of the 4th USENIX Windows Systems Symp. Washington: USENIX, 2000. 13−24.

[7] Hunt JJ, Vo KP, Tichy WF. Delta algorithms: An empirical analysis. ACM Trans. on Software Engineering and Methodology, 1998, 7(2):192−214. [doi: 10.1145/279310.279321]

[8] Quinlan S, Dorward S. Venti: A new approach to archival storage. In: Proc. of the 2002 Conf. on File and Storage Technologies. Monterey: USENIX Association, 2002. 89−101.

[9] Kulkarni P, Douglis F, LaVoie JD, Tracey JM. Redundancy elimination within large collections of files. In: Proc. of the USENIX Annual Technical Conf. Boston: USENIX Association, 2004. 59−72.

[10] Rhea S, Cox R, Pesterev A. Fast, inexpensive content-addressed storage in foundation. In: Proc. of the USENIX 2008 Annual Technical Conf. on Annual Technical Conf. Boston: USENIX Association, USENIX Association, 2008. 143−156.

[11] Lillibridge M, Eshghi K, Bhagwat D, Deolalikar V, Trezis G, Camble P. Sparse indexing: Large scale, inline deduplication using sampling and locality. In: Proc. of the 7th Conf. on File and Storage Technologies. San Francisco: USENIX Association, 2009. 111−123.

[12] Dubnicki C, Gryz L, Heldt L, Heldt L, Kaczmarczyk M., Kilian W, Strzelczak P, Szczepkowski J, Ungureanu C, Welnicki M. HYDRAstor: A scalable secondary storage. In: Proc. of the 7th Conf. on File and Storage Technologies. San Francisco: USENIX Association, 2009. 197−210.

[13] Ungureanu C, Atkin B, Aranya A, Gokhale S, Rago S, Calkowski G, Dubnicki C, Bohra A. HydraFS: A high-throughput file system for the HYDRAstor content-addressable storage system. In: Proc. of the 8th Conf. on File and Storage Technologies. San Jose: USENIX Association, 2010. 225−238.

[14] Zhu B, Li K, Patterson RH. Avoiding the disk bottleneck in the data domain deduplication file system. In: Proc. of the 6th Conf. on File and Storage Technologies. San Jose: USENIX Association, 2009. 1−14.

[15] Bobbarjung DR, Jagannathan S, Dubnicki C. Improving duplicate elimination in storage systems. ACM Trans. on Storage (TOS), 2006,2(4):424−448. [doi: 10.1145/1210596.1210599]

[16] Tridgell A. Efficient algorithms for sorting and synchronization [Ph.D. Thesis]. Canberra: Australian National University, 1999.

[17] Muthitacharoen A, Chen B, Mazieres D. A low-bandwidth network file system. ACM SIGOPS Operating Systems Review, 2001, 35(5):174−187. [doi: 10.1145/502059.502052]

[18] Rabin MO. Fingerprinting by random polynomials. Technical Report, TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[19] Rivest R. The MD5 message-digest algorithm. 1992. http://tools.ietf.org/html/rfc1321

[20] Eastlake D, Jones P. US secure hash algorithm 1 (SHA1). 2001. http://www.rfc-editor.org/info/rfc3174

[21] Policroniades C, Pratt I. Alternatives for detecting redundancy in storage systems data. In: Proc. of the USENIX 2004 Annual Technical Conf. Boston: USENIX Association, 2004. 73−86.

[22] Bloom BH. Space/Time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970,13(7):422−426. [doi: 10.1145/362686.362692]

[23] Clements AT, Ahmad I, Vilayannur M, Li J. Decentralized deduplication in SAN cluster file systems. In: Proc. of the 2009 Conf. on USENIX Annual Technical. San Diego: USENIX Association, 2009. 101−114.

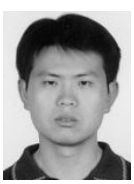
[24] Zhang W, Tang H, Jiang H, Yang T, Li X, Zeng Y. Multi-Level selective deduplication for VM snapshots in cloud storage. In: Proc. of the 5th IEEE Int'l Conf. on Cloud Computing (CLOUD). IEEE, 2012. 550−557. [doi: 10.1109/CLOUD.2012.78]

附中文参考文献:

[3] 敖莉,舒继武,李明强.重复数据删除技术.软件学报,2010,21(5):916−929. http://www.jos.org.cn/1000-9825/3761.htm [doi: 10.3724/SP.J.1001.2010.03761]

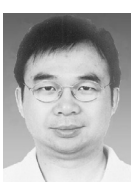
徐继伟(1985－),男,山东潍坊人,博士生,主要研究领域为网络分布式计算,软件工程.

钟华(1971－),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.



张文博(1976－),男,博士,研究员,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.



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.