软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(6):1875−1888 [doi: 10.13328/j.cnki.jos.005816]

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

基于分片复用的多版本容器镜像加载方法*

陆志刚[1,2,3], 徐继伟[1], 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: 容器将应用和支持软件、库文件等封装为镜像,通过发布新版本镜像实现应用升级,导致不同版本之间存在大量相同数据.镜像加载消耗大量时间,使容器启动时间从毫秒级延迟为秒级甚至是分钟级.复用不同版本之间的相同数据,有利于减少容器加载时间.当前,容器镜像采用继承和分层加载机制,有效实现了支持软件、库文件等数据的复用,但对于应用内部数据还没有一种可靠的复用机制.提出一种基于分片复用的多版本容器镜像加载方法,通过复用不同版本镜像之间的相同数据,提升镜像加载效率.方法的核心思想是:利用边界匹配数据块切分方法将容器镜像切分为细粒度数据块,将数据块哈希值作为唯一标识指纹,借助 B-树搜索重复指纹判断重复数据块,减少数据传输.实验结果表明,该方法可以提高 5.8X 以上容器镜像加载速度.

关键词: 容器;docker;镜像;重复数据删除

中图法分类号: TP316

中文引用格式: 陆志刚,徐继伟,黄涛.基于分片复用的多版本容器镜像加载方法.软件学报,2020,31(6):1875−1888. http://www.jos.org.cn/1000-9825/5816.htm

英文引用格式: Lu ZG, Xu JW, Huang T. Container image deduplication method based on chunking reuse of multi-versions. Ruan Jian Xue Bao/Journal of Software, 2020,31(6):1875−1888 (in Chinese). http://www.jos.org.cn/1000-9825/5816.htm

Container Image Deduplication Method Based on Chunking Reuse of Multi-versions

LU Zhi-Gang[1,2,3], XU Ji-Wei[1], HUANG Tao[1,2,3]

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

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

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

Abstract: Container encapsulates the application, the supporting software, and the operating system libraries as an image. The application is updated through publishing a newer image version. That would lead a certain degree of duplications between the neighboring versions. The loading process of container image is time-consuming and delays the starting time of a container from milliseconds to seconds or minutes. Reusing the same data of previous versions can help to reduce the loading time. The layered loading and inheritance features adopted by container can help to reuse the supporting software and the operating system libraries effectively in image loading. However, reusing the application data is currently not supported. This study proposed a container image loading methodology based on chunking reuse of older versions to improve the image loading performance. A boundary matching based chunking method was used to divide the image layers into fine-gained data chunk, the chunk hash value was used as the unique identification fingerprint. The B-tree was used to find the same blocks and the same blocks were reused to speed up the loading process. Experimental results show that the proposed method can improve 5.8X container image loading speed.

* 基金项目: 国家重点研发计划(2017YFC0804407); 国家自然科学基金(61602454, 61872344); 北京市自然科学基金(4182070)

Foundation item: National Key Research and Development Program of China (2017YFC0804407); National Natural Science Foundation of China (61602454); Beijing Nature Science Foundation (4182070)

收稿时间: 2017-09-04; 修改时间: 2018-05-22, 2018-09-08; 采用时间: 2019-01-15

Key words: container; docker; image; deduplication

容器(container)技术已经成为一种被广泛认可的服务器资源共享技术[1].与虚拟机相比,容器具有轻量级的特点,其快速启动、低系统资源消耗等优点使容器技术在弹性云平台和自动运维系统方面有着很好的应用前景.国内外主流云平台(如亚马逊 AWS、阿里云等)都相继开始支持 Docker 容器部署,各大、中、小型企业数据中心也正致力于采用 Docker 容器进行应用部署.Docker 容器将应用封装到镜像(image)中,通过运行镜像创建应用实例.由于镜像在镜像库(repository)中集中存储,将镜像从集中存储加载到本地的过程需要耗费大量的时间,这将导致镜像实例创建时间由毫秒级延迟为分钟级,严重制约了应用在面对突发高负载时的处理能力,影响应用服务质量.虽然 Docker 镜像采用了分层加载的机制,但是由于受到应用版本多样性等因素影响,在不同应用版本镜像之间依然存在重复存储和重复加载的现象.复用已加载的数据可以有效提高镜像加载速度,对提高应用的弹性扩展能力具有重要意义.

重复数据查找技术是数据去冗余中的一种常用技术,广泛应用于数据存储和数据迁移中[2,3],其核心思想是基于内容寻址进行数据分片复用.将数据去冗余技术应用到容器镜像存储和加载过程,可以有效提高容器镜像存储设备利用率、提高容器镜像加载速度.充分利用旧版本镜像中的相同数据,可以有效提高新版本镜像的加载速度.

本文分析了 Docker 容器镜像分层存储和加载机制,提出了一种基于分片复用的容器镜像加载方法.方法通过增加元数据信息实现对镜像数据的有效组织,使已有镜像数据具有可复用性,从而达到提高镜像加载效率的目的.本文的贡献如下:

(1) 首次提出了多版本容器镜像加载过程中存在数据重复加载问题,并对其产生原因进行分析,指出原生加载策略存在的不足;

(2) 提出一种基于重复数据删除技术的镜像数据分片复用方法,有效复用不同版本容器镜像的相同数据,降低镜像加载过程中的网络传输数据量,提高镜像加载效率;

(3) 分析镜像加载策略,针对不同场景下的镜像加载需求,自动选择最优的镜像加载策略,减少不必要的操作开销;

(4) 采用模拟实验,通过设置不同实验参数进行多组实验,验证本文提出方法的有效性.

本文第 1 节介绍本文的研究背景和研究动机,主要介绍 Docker 容器和镜像的组成结构,分析 Docker 镜像加载过程中存在的问题.第 2 节提出一种基于重复数据删除技术的镜像加载方法,减少镜像加载过程中的网络传输.第 3 节通过实验验证本文提出方法的有效性和可行性.第 4 节系统地总结和分析对比相关工作.第 5 节对本文工作进行总结.

1 研究背景与动机

1.1 Docker容器及镜像

容器是一种进程级虚拟化方案,容器镜像用于容器的数据存储并可以被实例化为容器实例.容器实例在运行过程中会复用主机的操作系统内核,因此,容器镜像只负责封装操作系统库文件、中间件、应用及其配置等内容.为了提高存储和加载效率,Docker 镜像由一组只读的文件层组成,每个文件层均为一个单独的文件系统.在镜像实例化过程中,不同文件层按顺序叠加组成了容器实例的根文件系统[4].如图 1 所示,以 Ubuntu 17.04 容器镜像为例,该镜像包含 5 个不同的文件层,每层所包含内容为该层文件与上层文件的差异.Docker 镜像在实例化时,所有的镜像文件层设为只读层,在镜像文件层上面添加一个可读写的层,该层通常被称为容器层.容器实例运行过程中,所有的文件系统修改操作(如写文件、修改文件、删除文件等)均发生在容器层.

Docker 容器和 Docker 镜像的最大区别,就在于最上层的这个可读写的文件层.当容器删除时,最上层的容器层会随之删除,而镜像文件层则不受影响.由于每个容器有不同的容器层并且对于容器的修改都存储于相应的容器层当中,多个容器可以通过共享的方式访问相同的镜像文件层,同时具有自己的数据状态.Docker 存储驱

动采用写时复制(copy-on-write)技术实现上述功能.Docker 容器通过特定的操作可以转化为新的镜像,即:将可写的容器层设为镜像的最上层,并将其状态置为只读状态.

由此可见,镜像和容器的相互转化关系是,容器可以直接转化为镜像,而镜像不能直接转化为容器,只能通过在最上层添加一个可读写文件层实例化为容器.

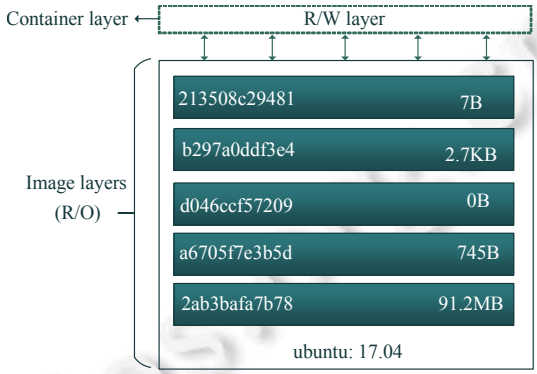

Fig.1 Docker container layers

图 1 Docker 容器文件层

1.2 镜像存储与加载过程中存在的问题

在分布式环境中,同一镜像对应的容器实例会运行在不同的主机上,Docker 采用服务器/客户端的模式来存储和分发镜像.用户一般采用 Docker 镜像库(docker registry)来集中存储镜像.Docker 镜像库是 Docker 生态系统中的重要组件,是一个存储和内容分发系统,存储已命名的镜像和用不同 tag 标记的可用版本.

镜像加载是指镜像从镜像库加载到运行主机的过程.由于 Docker 镜像具有分层的特性,且每层均以该层哈希值命名,在加载过程中,只需加载对应哈希值不存在的文件层.Docker 镜像加载过程可分为以下几步:1. 客户端将镜像名称和版本号发送到相应的镜像库服务器;2. 服务器根据名称和版本号查找对应的镜像,并将组成该镜像的不同文件层的相关信息(如哈希值)返回客户端;3. 客户端根据服务器返回的文件层信息,在本地查找相同文件层;4. 如果客户端在本地没有找到某文件层,则向服务器发送加载该文件层请求;5. 服务器端通过网络将步骤 4 中请求的文件层发送到客户端;6. 客户端根据步骤 2 中返回的信息将各文件层组织为镜像.因此,Docker 镜像加载本身是一种文件层去冗余的加载过程.图 2 所示为 Vote App 从 1.0.0 版本升级为 1.1.0 版本过程,其中,L1~L4 文件层均没有发生变化,而 L5 文件层发生变化.

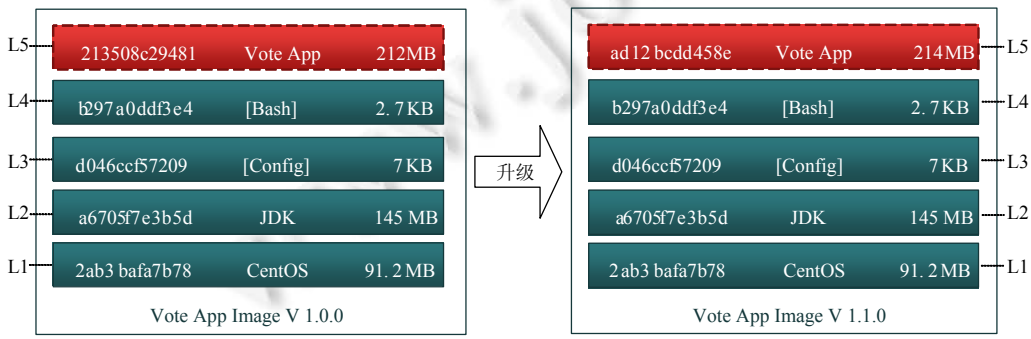

Fig.2 Docker image update diagram

图 2 Docker 镜像升级示意图

按照镜像加载策略,在升级过程中,L1~L4 文件层不需要重新加载,而 L5 文件层需要全部重新加载.L5 文件

层数据量大小约占整个镜像的 47%,而两个版本的数据改变量不到 1%.于是,我们可以得出一个结论:虽然当前的镜像加载方式采用了去冗余加载的策略,但是在特定的情况下,仍然需要加载大量冗余数据.主要原因是:当前去冗余加载是以文件层为粒度进行去冗余,而更细粒度的数据情况改变无法获得.Docker 设计者的本意是鼓励用户尽量将应用微服务化,减少每个 Docker 容器数据量,然而在实际使用过程中,大量应用开发者仍按照旧有思路开发应用,使应用对应的镜像文件层数据量不断变大.近年来,随着 DevOps 技术的不断发展和进步,应用开发速度和升级频率逐渐加快,每天 10 次的应用发布频率已成为很多互联网公司的标准配置.因此,在基于 Docker 的应用部署架构中,减少应用更新时的网络数据传输量可以加快应用更新速度、提高服务质量.

综上可知,Docker 镜像升级过程中可能存在大量数据重复加载的问题,解决这个问题对提升服务质量具有重要意义.

2 基于重复数据删除技术的镜像数据分片复用加载方法

本文的工作目标是:通过复用重复数据来降低镜像加载过程时间消耗,提高镜像加载效率.为达到这个目的,我们首先需要发现并识别镜像中的重复数据.重复数据识别可包括数据块级重复数据识别[5]和文件级重复数据识别[6],其主要区别在于:数据块级重复数据识别需要对数据进行细粒度切分后再进行比较,而文件级重复数据识别则直接对文件进行比较.因而数据块级重复数据识别更加适用于数据流和少量变化频繁的大文件,而文件级重复数据识别则适用于小文件或变化较少的大文件.容器镜像加载过程中镜像文件层以一个整体进行流式传输,因此本文中采用数据块级重复数据识别方法.重复数据识别方法原理见公式(1).

已知存在数据块哈希值集合 S,对于任意数据块 b,其哈希值 $h=H(b)$,则有:

$$V=\begin{cases}1, & \text{if } h\in S\\ 0, & \text{if } h\notin S\end{cases} \tag{1}$$

其中,输出结果 1 代表数据块为重复数据;输出结果 0 代表数据块为非重复数据块.此时,则有:

$$S=S\cup\{H(b)\} \tag{2}$$

下面,我们将具体介绍方法细节.

2.1 边界匹配数据块切分

如图 3(a)所示:由于存在对文件层数据的插入、修改、删除等操作,会导致边界漂移问题的出现[7],固定长度分块算法无法满足准确切分冗余数据块的需求.为此,我们需要一种能够有效识别数据块边界的可变长度数据块切分方法,如图 3(b)所示.为实现上述目标,我们首先采用一种与边界相关的指纹算法计算滑动窗口内数据指纹,再通过特定条件加以遴选,确定边界符合条件的数据块.

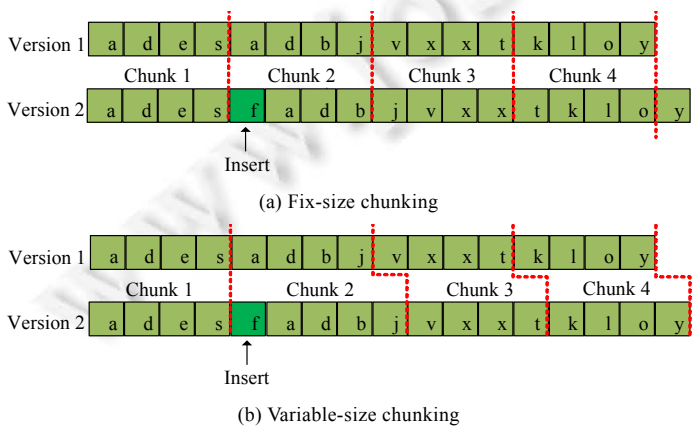

Fig.3 Data chunking diagram

图 3 数据块切分示意图

本文中我们采用 Rabin 指纹算法[8].Rabin 指纹算法是一种边界相关的指纹算法,其实现如下:给定 n 位的数据 $m_0,m_1,\ldots,m_{n-1}$,我们将其视作在有限域上度为 $n-1$ 的多项式:

$$f(x)=m_0+m_1x+\ldots+m_{n-1}x^{n-1}.$$

选择一个度为 k 的不可约多项式 $p(x)$,数据指纹通过公式(3)计算:

$$r(x)=\frac{f(x)}{p(x)} \tag{3}$$

给定整数 m,n,若满足公式(4),则视为满足边界条件:

$$\frac{r(x)}{m}=n \tag{4}$$

基于边界匹配的数据块切分方法如图 4 所示.我们假定滑动大小为 W,移动步长为 s,最大数据块大小为 M.从位置 0 开始,通过公式(4)判断滑动窗口内数据[0,W]是否满足边界条件:若满足,则将进行切分;若不满足,则按步长移动滑动窗口至新位置 ns,直至滑动窗口内数据满足边界条件,则将[0,ns]数据和[ns,$ns+W$]数据分别切分为数据块.移动步长超过最大数据块大小 M,则将[0,M]数据切分为数据块,并将位置 M 重置为起始位置 0.

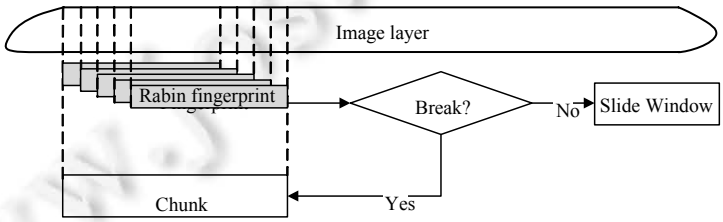

Fig.4 Boundary matching based chunking method diagram

图 4 基于边界匹配的数据块切分方法示意图

2.2 B-树数据指纹搜索

我们通过比较数据块指纹是否相同,来判断数据块是否为重复数据.由于 Rabin 指纹算法散列值的空间较小,容易产生哈希冲突,为避免哈希冲突,我们使用 SHA-1 算法重新计算每个数据块的指纹,并将该数据块对应指纹存储到指纹库中.容器镜像切分后将产生大量数据指纹,以数据块平均大小为 4KB 计算,1TB 镜像数据切分后,将产生 2.7×10^8 条数据.在数据条目多的情况下,散列表将出现大量哈希冲突,严重影响搜索效率.因此,我们选择搜索复杂度为 $O[\log_2N]$的 B-树结构进行存储.另外,散列表的一般利用率约为 50%,由于指纹库数据量较大,会浪费大量的存储空间.

B-树是平衡多路查找树.树节点结构设计如图 5 所示,节点记录了数据块对应的 SHA-1 值、数据块存储的物理地址(physical address)、数据块相对于物理地址的偏移量(offset)和数据块长度等信息(length).其中,SHA-1 值作为节点的关键字(key),用于数据指纹的查找和比对.一棵 m 阶的 B-树中,每个结点最多含有 m 个子节点($m\geqslant2$),除根结点和叶子结点外,其他每个结点至少有[$m-2$]个孩子,每个非叶子结点中包含有 n 个关键字信息和 $n+1$ 个指针信息:($n,P_0,P_1,K_1,\ldots,P_n,K_n$).其中,$([m/2]-1)\leqslant n\leqslant(m-1)$;$K_i(i=1,\ldots,n)$为关键字,且满足 $K_{i1}<K_i;P_i$ 为指向子树指针,且指针 P_{i1} 指向子树所有结点的关键字均小于 K_i,大于 K_{i1}.

SHA-1 value	Physical address	Offset	Length

Fig.5 Information of B-tree node

图 5 B-树节点信息

由于指纹数据量大,无法全部加载到内存中,降低指纹搜索复杂度可以有效减少磁盘 I/O,提高搜索效率.我们取得数据块指纹之后,在基于 B-树的指纹库中进行搜索,其搜索过程等价于在指纹全集中做一次二分查找.查找过程中,如果指纹存在,则该数据块为重复数据块,可不用重复加载;如果指纹不存在,则该数据块为非重复数

据块,需要进行加载.

2.3 策略决策

对镜像数据进行分片处理,可以降低重复数据加载.但同时,由于增加了一些额外的系统开销(如数据块切分、指纹计算、指纹搜索等),这在一定程度上可能会影响系统效率[9].考虑两种极端情况:一是当重复数据为 0 的情况,系统非但没有达到减少数据加载量的目的,还白白增加了额外系统开销;二是当重复数据为 100%的情况下,额外系统开销没有发生变化,而网络数据传输量则减少为 0.大多数时候系统处于这两种极端情况之间,因此我们需要在基于重复数据删除技术的镜像数据分片复用加载方法和原生镜像加载方法之间做权衡.

我们就利用决策树对镜像加载方法进行选择.容器镜像集中存储在容器库(repository)中,宿主机发送镜像加载请求后,镜像库服务器根据请求内容进行反馈.为应用本文提出的方法,我们对传统的镜像加载流程进行改造,如图 6 所示.在宿主机和镜像库服务器上分别放置一个 Agent 用于监测和计算系统当前各资源使用状态和镜像情况等信息,用于作为决策树判断的依据.发起请求时,宿主机将 Agent 收集到的信息一并发送至镜像库服务器,镜像库服务器将宿主机发送来的信息和本机 Agent 上收集的信息发送给决策树,由决策树判定采用何种加载方法.确定之后,系统将根据所确定的方法传输相应的数据到宿主机,宿主机根据接收到的数据类型重组镜像,从而完成整个加载流程.

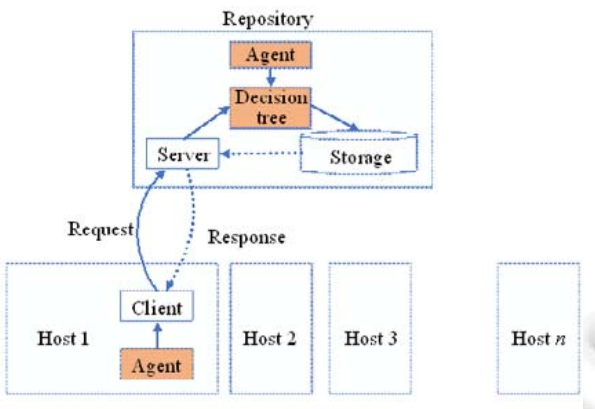

Fig.6 Decision tree based image load diagram

图 6 基于决策树的镜像加载示意图

决策树构建需要选择对加载过程产生影响的因子作为特征值,我们从以下几个方面选取.

1) 数据方面.数据方面影响因子包括数据大小(S)和去冗余率(r),镜像加载过程中的网络传输数据量可以根据以上两个因子计算得出;
2) 网络方面.镜像通过网络从镜像库加载到宿主机,因此选取网络传输速度(n_s)作为决策树特征值;
3) 磁盘方面.镜像库磁盘读取速度(R)和宿主机端磁盘写速度(W)影响镜像数据加载过程,因此选择这两个属性作为特征值;
4) 内存方面.镜像传输和 B-树查找都是内存密集型运算,我们使用可用内存大小(Ma)和指纹库数据大小(A)之间的比值作为决策树特征值;
5) CPU 方面.CPU 在数据块指纹计算方面发挥重要作用,直接影响加载效率,因此选择 CPU 利用率、payload 和 real-time 这 3 个指标作为特征值;
6) 其他方面.在分布式系统中,系统负载经常发生变化,因此我们需要加以考虑.由于系统负载难以刻画,而系统负载通常随时间变化产生周期性变化,因此我们采用时间(t)作为特征值.

假设训练决策树的样本数据集为 D,决策树的输出有两种,分别为基于重复数据删除的镜像加载和原生镜像加载方法.在构建决策树时,选择某个特征值作为树的节点.根据信息论,计算出该数据中的信息熵:

$$Info(D)=-\sum_{i=1}^{2}P_i\log_2(P_i) \tag{5}$$

其中,P_i表示类别为 i 样本数量占所有样本的比例.在特征 A 作用下的信息熵计算公式如下:

$$Info_A(D)=-\sum_{j=1}^{k}\frac{|D_j|}{|D|}\times Info(D_j) \tag{6}$$

其中,k 表示样本 D 分为 k 个部分.数据集 D 在特征 A 的作用下信息熵减少的值为信息增益,计算公式如下:

$$Gain(A)=Info(D)-Info_A(D) \tag{7}$$

利用训练样本数据集,根据上述公式即可构建决策树.

2.4 方法开销

我们在原生方法的基础上增加了数据块级别重复数据删除方法和基于决策树的策略决策方法,因此相比原生方法会产生额外的开销.本节我们分别分析两种方法对系统性能产生的影响.

数据块级别重复数据删除方法主要由 3 个子任务构成.

1) 数据分块及指纹计算.将镜像划分为不同数据块并计算数据块指纹.该操作为 CPU 敏感型操作,在服务器和客户端分别执行;
2) 数据块指纹查找.在指纹库中查找数据块指纹及数据块存储地址,用于判断该数据块是否已存在和确定数据块的读取位置.该操作为内存敏感型操作,在服务器和客户端分别执行.由于服务器端指纹库远大于客户端指纹库,因此资源消耗更多;
3) 网络传输.将数据从服务器发送到客户端,该操作为网络敏感型操作.由于只需要发送发生变化的数据块,相比镜像整体传输网络资源占用会变小.

从以上分析中可以看出:采用数据块级别重复数据删除方法进行镜像加载,在降低网络负担的同时增加了额外的操作.本文中,我们主要关注任务的完成时间,所以我们从任务时间成本出发,提出了基于决策树的策略决策方法,用于对特定的镜像选择合适的加载策略.

基于决策树的策略决策方法的建模过程为离线操作,其开销不在本文讨论范围内之内.在运行过程中,其主要开销应为 Agent 对系统资源使用状态监测,然而资源监控是数据中心运维的一项重要功能,系统只需要调用运维监控系统的数据即可完成数据指标采集.在结果计算过程中,决策时间为毫秒级,在本文中,我们视为可以忽略.

3 实验验证

我们在 Docker 1.9.1 版本和 Docker Register V2 版本上实现了基于重复数据删除技术的镜像数据分片复用加载方法,本节我们通过实验验证方法的有效性和稳定性.

3.1 实验设置

通过实验,我们首先验证边界匹配数据块切分方法去冗余的有效性,然后将基于重复数据删除技术的镜像加载方法与原生镜像加载方法进行对比,最后验证数据规模不断扩大情况下方法的稳定性.

实验中,我们采用 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.其中,4 台作为宿主机,一台作为镜像库服务器.镜像库存储采用 10TB 空间的存储集群.所有设备通过千兆网络设备进行连接.

我们从 OnceCloud[10]云平台中选取大小不同(100MB~500MB)的 10 种不同类型的镜像,并通过模拟升级的方式进行独立演化.针对每个镜像我们模拟了 100 个升级版本,版本间数据变化率设置为 0.1%~10%,其中,新增数据和修改数据比例为 1:1.实验中所涉及的数据变化率和重复数据率,均指镜像最上层文件发生的数据变化.实验用总数据量约为 200GB.数据块切分时,滑动窗口大小设为 4KB,移动步长设为 512B,最大数据块大小为 8KB.宿主机上采用 Device Mapper 作为 Docker 存储驱动.

3.2 重复数据识别率

重复数据识别率是衡量数据块切分方法有效性的重要指标,高重复数据识别率表示数据块切分方法更加有效,在镜像加载过程中可以更大限度减少网络传输开销.我们采用边界匹配数据块切分方法对实验数据集进行切分并查找重复数据,以镜像类型和数据变化率两个维度对切分结果进行统计,统计结果分别如图 7 和图 8 所示.

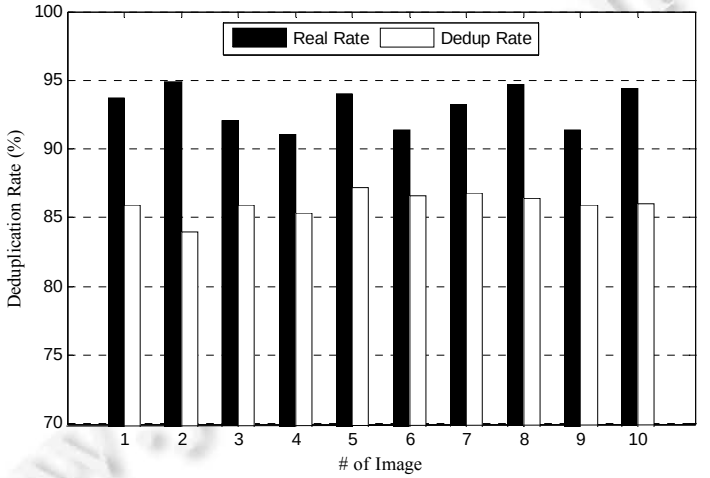

Fig.7 Real duplication rate vs. deduplication rate

图 7 实际重复数据率与重复数据识别率对比

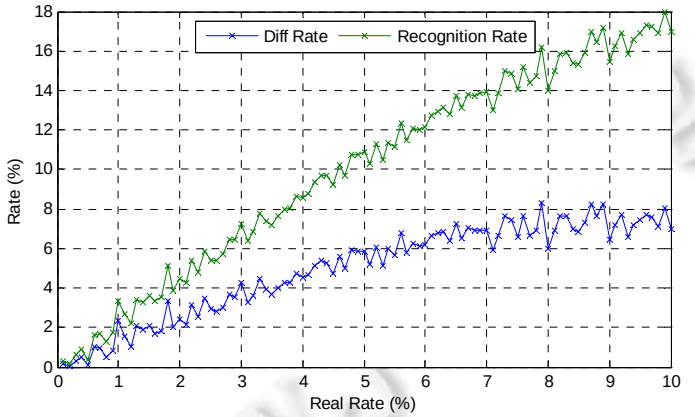

Fig.8 Relationship between real data change rate and recognition data change rate

图 8 实际数据变化率与识别数据变化率对比

图 7 所示为按镜像类型进行统计的结果,其中,横轴代表 10 个不同类型的版本,纵轴代表冗余数据率.图中黑色图例(real rate)表示同种镜像不同版本的真实重复数据率的平均值,白色图例(dedup rate)表示通过数据块切分后重复数据识别率的平均值.从图中我们可以看出:真实重复数据率与重复数据识别率的最大相差 11.4%(图中第 2 个镜像),最小为 5.3%(图中第 6 个镜像),平均相差 7.6%.这说明边界匹配数据块切分方法能够很好的识别重复数据,可以有效应用于镜像分片复用加载.

图 8 所示为按镜像数据更新率进行统计的结果,其中,横轴表示真实数据变化率,recognition rate 图例表示重复数据识别方法识别出的数据变化率,diff rate 图例表示识别数据变化率与真实数据变化率之间的差值.从图

中我们可以看出:(1) 随着数据变化率不断增大,识别数据变化率也随之增加,其识别数据变化率始终大于真实数据变化率,这是因为我们所用的镜像不存在内部重复数据[11],且每次版本更新与之前,版本没有相同的数据块;(2) 随着数据变化率不断增大,识别数据变化率与真实数据变化率之间的差值也在不断增大.但当数据变化率增大到一定程度时,该值变化趋于稳定,其波动范围在 6%~8%之间.实验结果表明:边界匹配数据块切分方法能在 5.3%~11.4%误差范围内识别重复数据,且受到数据变化率的影响较小,证明方法具有较好的有效性和稳定性.

3.3 加载效率

我们通过实际系统测试对比基于重复数据分片复用加载方法和原生加载方法的加载效率,实验结果以镜像类型和数据变化率两个维度进行统计,结果分别如图 9 和图 10 所示.我们将原生加载方法的加载速度作为基准,考察基于重复数据分片复用加载方法的加载速度与原生加载方法加载速度的比值,即方法的加速比,图中用 Speed Factor 表示.

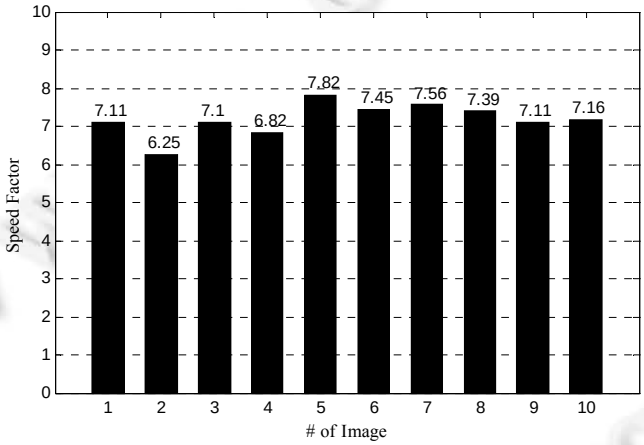

Fig.9 Average speed factor of different image types

图 9 不同类型镜像平均加载比

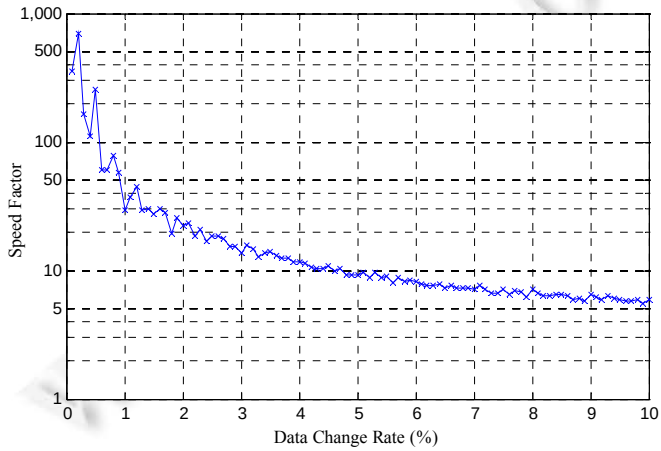

Fig.10 Relationship between real data change rate and recognition data change rate

图 10 实际数据变化率与识别数据变化率对比

图 9 所示为不同镜像类型各版本的平均加速比,其中,横轴代表 10 个不同类型的版本,纵轴代表方法的加速比.从图中我们可以看出:基于重复数据分片复用加载方法平均可以提高 7X 左右的加载速度,最高平均可提高

7.82X 的加载速度,最低平均可提高 6.25X 的加载速度.

图 10 所示为不同镜像更新率下方法的加速比,其中,横轴代表不同的镜像更新率,纵轴代表方法加速比.从图中我们可以看出:当镜像更新率小于 0.2%时,方法可以达到 300X~700X 的加速比;当镜像更新率在 4%左右时,方法可以达到 10X 以上的加速比;当镜像更新率为 10%时,方法加速比约为 5.8X.方法的加速比随着镜像更新率的不断提高而逐渐减小.这是因为镜像更新率越高,镜像不同版本之间的重复数据越少,可复用的数据越少,镜像加载时通过网络传输的数据就越多.

实验结果表明,基于重复数据分片复用加载方法可以有效提高镜像加载效率.

3.4 指纹查找时间

在前面的实验中,我们假设内存足够大可以加载整个数据块指纹库,但在实际系统中,数据块指纹库会随着镜像的增多而不断增大.我们采用 SHA-1 散列值作为数据块指纹,每个数据块指纹大小为 20B.按平均每个数据块大小为 4KB 计算,20GB 非重复数据对应的指纹库大小为 100MB.我们通过实验的方式验证当可用内存大小超出指纹库大小时,指纹查找的效率.我们将可用内存大小设置为 100MB,数据块指纹库大小从 100MB 递增到 500MB.我们选取一个大小为 200MB 的容器镜像的指纹序列(约为 5 000 条指纹)进行指纹搜索操作,记录不同指纹库大小时查询操作所需要的时间.

实验结果如图 11 所示,其中,横轴代表指纹库大小,纵轴代表查询操作所需要的时间.当指纹库大小为 100MB,内存刚好容纳可以所有数据指纹时,查询操作所需要的时间为 0.2s.随着指纹库逐渐增大,查询操作所需要的时间也随之增加.当指纹库大小为 500MB 时,查询操作所需要的时间约为 8s.实验结果表明:当可用内存大于指纹库时,指纹查找时间可用忽略不计;但当可用内存小于指纹库时,指纹查找时间显著增加,可能会影响方法的效率.若指纹查找时间超过分片复用加载方法所带来的的时间收益时,本文方法反而会使加载过程变慢.

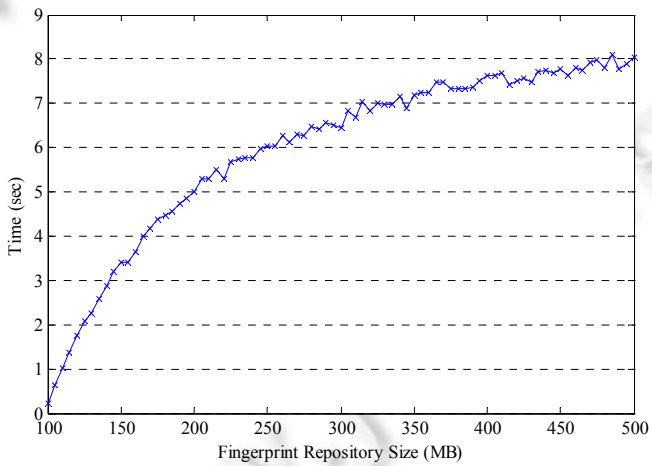

Fig.11 Relationship between fingerprint repository size and search time

图 11 指纹库大小与查找时间的关系

3.5 策略决策比例

图 12 所示为实验过程中经过策略决策后所采用的两种加载方法所占的比例,其中,横轴代表 10 个不同类型的镜像版本,纵轴代表两种方法在经过决策后所占在比例.图中黑色图例代表采用基于重复数据删除技术加载方法所占的比例,白色图例代表采用原生镜像加载方法所占的比例.从图中我们可以看出:本文提出的基于重复数据删除技术的镜像加载方法所占最小的比例约为 86.2%,最大比例约为 88.8%,平均比例约为 87.8%;而原生镜像加载方法所占最小比例约为 11.2%,最大比例约为 13.8%,平均比例约为 12.2%.

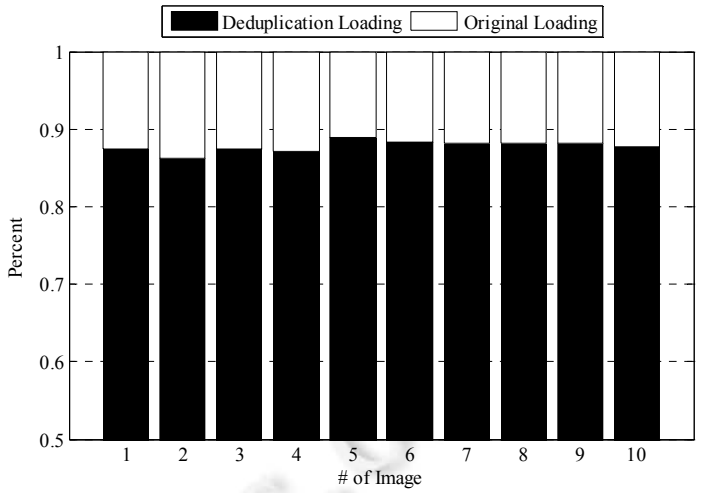

Fig.12 Ratio of the two loading methods

图 12 两种加载方法所占比例

实验结果表明:本文提出的基于重复数据删除技术的镜像加载方法可以有效应对多版本镜像升级这一使用场景,提高升级版本镜像的加载效率.

4 相关工作

如何提高 Docker 镜像加载效率,已成为一个热点研究问题[12].为解决该问题,研究人员针对不同的场景提出了不同的解决方案.Wang 等人[13]针对大规模镜像加载并发冲突问题,提出了基于 P2P 的镜像加载解决方案.Liang 等人[14]将 P2P 加载方法和原生镜像加载方法相结合,提出了一种混合镜像加载方法.基于 P2P 的镜像加载方法充分利用了节点间的网络带宽,提高了镜像加速效率.Du 等人[15]使用 Ceph 作为共享存储,利用 Ceph 提供的快照克隆等功能,来提高不同节点之间镜像加载效率.以上方法提高了特定情况下的镜像加载效率,但镜像加载所需要的网络数据传输总量没有变化.Harter 等人[16]将共享存储和重复数据删除技术相结合,提出了 Slacker 平台.Slacker 基于共享的 NFS 文件系统,在所有节点和 Registry 之间共享镜像.这种懒加载方式可以大大提高镜像加载效率,然而为了便于快照和克隆,Slacker 需要将 Docker 镜像扁平化为单独一层,这与 Docker 方便创建和更新镜像的设计理念相冲突.本文仅针对镜像升级这一特殊却又广泛存在的使用场景,提出利用重复数据删除技术进行镜像加载,同时考虑到方法开销与收益的平衡,又提出利用决策树进行方法选择.与上述方法相比,本文方法在镜像升级这一应用场景下,可以有效降低网络数据传输,提高镜像加载效率.

重复数据删除技术[17,18]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数据,保证相同的数据只存储一份,以达到降低存储空间和传输开销的目的.按照去冗余数据单元粒度划分,可以将去冗余方法分为文件级别去冗余[6]和数据块级别去冗余[18]:文件级别去冗余首先计算文件的哈希值作为文件指纹,然后与指纹库中的哈希值进行比较判断文件是否重复;数据块级别去冗余首先将数据切分粒度较小的数据块,计算每个数据块的哈希值,与指纹库中的哈希值进行比较,来判断数据块是否重复.数据块切分方法可以分为两种:一种是固定长度切分方法[5,18,19],一种是可变长度切分方法[20−22].本文采用的是可变长度分块方法.Policroniades 等人[23]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.可见,可变长度切分去冗余将大大提高操作的时间复杂度.

重复数据删除技术可以应用于数据备份、数据迁移、文件系统管理等多种不同场景.Fu 等人提出的应用感知的数据去冗余技术[24]应用于云备份服务中,主要解决数据传输效率问题.该方法通过特定的索引机制,将全部索引划分为若干相互独立且具有典型应用特性的组织形式,达到缓解索引查找瓶颈的目的.基于语义的源端

重复数据删除技术[25]主要用来解决备份中的数据量传输过高导致备份时间窗口长的问题,该方法通过混合重复数据删除技术和语义层冗余数据消除技术来降低备份过程中的数据传输量.基于 Hadoop 重复数据删除方法[26,27]和基于固态硬盘的重复数据删除方法[28]可以用于在特定应用场景下加速重复数据删除过程.Zhang 等人将重复数据删除技术应用到 OpenStack 中,开发了虚拟机镜像管理系统 IM-dedup[29].该系统采用固定分块的技术将虚拟机镜像切分为大小相等的数据块,通过使用指纹验证是否重复数据块的方式,避免冗余数据块通过网络进行重复传输;同时,通过部署采用了去冗余技术的内核态文件系统来降低存储空间占用.Ng 等人开发了 LiveFS 系统实现镜像存储的实时去冗余[30],降低磁盘存储空间使用.分布式多层次选择去冗余[31]系统在 Inner-VM 去冗余阶段实现虚拟机内部数据块级重复数据删除,在 Cross-VM 去冗余阶段,针对不同类型的操作系统进行分组,对分组内的虚拟机镜像进行去冗余,因而可以利用较小的内存空间完成指纹查找过程.Zhang 等人[32−34]将重复数据删除技术应用于虚拟机迁移过程中,通过数据指纹发现相同内存页,减少迁移内存数据量.由于需要解决尖峰负载、即时失效、集群维护等问题,数据中心可能会面临虚拟机组迁移(gang migration)的问题.不同宿主机内存中可能存在大量相同的内存页,Deshprade 等人提出了集群范围内去冗余方案(GMGD)[35],方法通过一种冗余跟踪机制对不同虚拟机中的相同内存页进行跟踪,利用一种分布式协调机制阻止相同的数据块通过核心链路进行重新发送.

5 总 结

将重复数据删除技术用于容器镜像加载,可以提高容器镜像加载速度.容器镜像采用联合文件系统技术实现了可以实现文件层级别的数据复用.由于以文件层为单位进行划分粒度较粗,无法实现对同一文件层内部重复数据的复用.在镜像版本升级时,复用文件层内部数据可以有效提高镜像加载效率.我们提出一种基于重复数据删除技术的镜像数据分片复用加载方法提高原生容器镜像加载过程,利用边界匹配数据块切分方法和 B-树搜索发现重复数据块并加以复用.由于重复数据删除技术需要一定的资源开销,在实际使用中需要平衡方法的收益与开销,以达到最优效果.为此,我们利用决策树选择合适的加载方法.实验结果显示:我们的方法稳定有效,可以提高 5.8X 以上的镜像加载速度.在今后的工作中,我们将继续优化该方法,并在实际系统中检验方法的稳定性和有效性.

References:

[1] Merkel D. Docker: Lightweight linux containers for consistent development and deployment. Linux Journal, 2014.

[2] Keren J, Miller EL. The effectiveness of deduplication on virtual machine disk images. In: Proc. of the SYSTOR 2009: The Israeli Experimental Systems Conf. ACM, 2009.

[3] Jayaram KR, Peng CY, Zhang Z, Kim MK, Chen H, Lei H. An empirical analysis of similarity in virtual machine images. In: Proc. of the Middleware 2011 Industry Track Workshop. ACM, 2011.

[4] Docker docs. https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#images-and-layers

[5] 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.

[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] 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]

[8] Rabin MO. Fingerprinting by random polynomials. Tech Report TR-CSE-03-01. Center for Research in Computing Technology, Harvard University.

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

[10] Institute of Software Chinese Academic of Sciences. ONCECloud: Open network computing environment. 2017. https://github.com/oncecloud

[11] Zhang W, Yang T, Narayanasamy G, *et al*. Low-Cost data deduplication for virtual machine backup in cloud storage. In: Proc. of the 5th USENIX Conf. on Hot Topics in Storage and File Systems (HotStorage), Vol.13. 2013. 2–2.

[12] Ahmed A, Pierre G. Docker container deployment in fog computing infrastructures. In: Proc. of the IEEE EDGE 2018-IEEE Int'l Conf. on Edge Computing. IEEE, 2018. 1–8.

[13] Wang KJ, Yong Y, Ying L, *et al*. FID: A faster image distribution system for docker platform. In: Proc. of the 2017 IEEE 2nd Int'l Workshops on Foundations and Applications of Self* Systems (FAS*W). IEEE, 2017. 191–198.

[14] Liang M, Shen S, Li D, *et al*. HDID: An efficient hybrid docker image distribution system for datacenters. In: Proc. of the Software Engineering and Methodology for Emerging Domains. Singapore: Springer-Verlag, 2016. 179–194.

[15] Du L, Wo T, Yang R, *et al*. Cider: A rapid docker container deployment system through sharing network storage. In: Proc. of the 2017 IEEE 19th Int'l Conf. on High Performance Computing and Communications, IEEE 15th Int'l Conf. on Smart City, IEEE 3rd Int'l Conf. on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2017. 332–339.

[16] Harter T, Salmon B, Liu R, *et al*. Slacker: Fast distribution with lazy docker containers. FAST, 2016,16:181–195.

[17] Shin Y, Koo D, Hur J. A survey of secure data deduplication schemes for cloud storage systems. ACM Computing Surveys (CSUR), 2017,49(4):74.

[18] 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]

[19] Xu J, Zhang W, Zhang Z, *et al*. Clustering-Based acceleration for virtual machine image deduplication in the cloud environment. Journal of Systems and Software, 2016,121:144–156.

[20] Park D, Fan Z, Nam Y J, *et al*. A lookahead read cache: Improving read performance for deduplication backup storage. Journal of Computer Science and Technology, 2017,32(1):26̃40.

[21] Zhang Y, Feng D, Jiang H, *et al*. A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems. IEEE Trans. on Computers, 2017,66(2):199–211.

[22] Wu S, Li KC, Mao B, *et al*. Dac: Improving storage availability with deduplication-assisted cloud-of-clouds. Future Generation Computer Systems, 2017,74:190–198.

[23] Policroniades C, Pratt I. Alternatives for detecting redundancy in storage systems data. In: Proc. of the USENIX Annual Technical Conf. General Track, 2004. 73–86.

[24] Fu Y, Jian H, Xiao N, *et al*. AA-Dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment. In: Proc. of the 2011 IEEE Int'l Conf. on Cluster Computing (CLUSTER). IEEE, 2011. 112–120.

[25] Tan Y, Jiang H, Feng D, *et al*. SAM: A semantic-aware multi-tiered source de-duplication framework for cloud backup. In: Proc. of the 2010 39th Int'l Conf. on Parallel Processing (ICPP). IEEE, 2010. 614–623.

[26] Kolb L, Thor A, Rahm E. Dedoop: Efficient deduplication with hadoop. Proc. of the VLDB Endowment, 2012,5(12):1878–1881.

[27] Liu Q, Fu Y, Ni G, *et al*. Hadoop based scalable cluster deduplication for big data. In: Proc. of the 2016 IEEE 36th Int'l Conf. on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2016. 98–105.

[28] Mao B, Jiang H, Wu S, *et al*. SAR: SSD assisted restore optimization for deduplication-based storage systems in the cloud. In: Proc. of the 2012 IEEE 7th Int'l Conf. on Networking, Architecture and Storage (NAS). IEEE, 2012. 328–337.

[29] Zhang J, Han S, Wan J, *et al*. IM-Dedup: An image management system based on deduplication applied in DWSNs. Int'l Journal of Distributed Sensor Networks, 2013.

[30] Ng CH, Ma M, Wong TY, *et al*. Live deduplication storage of virtual machine images in an open-source cloud. In: Proc. of the 12th Int'l Middleware Conf. Int'l Federation for Information Processing, 2011. 80–99.

[31] Zhang W, Tang H, Jiang H, *et al*. Multi-Level selective deduplication for vm snapshots in cloud storage. In: Proc. of the 2012 IEEE 5th Int'l Conf. on Cloud Computing (CLOUD). IEEE, 2012. 550–557.

[32] Zhang X, Huo Z, Ma J, *et al*. Exploiting data deduplication to accelerate live virtual machine migration. In: Proc. of the 2010 IEEE Int'l Conf. on Cluster Computing (CLUSTER). IEEE, 2010. 88–96.

[33] Nathan S, Bellur U, Kulkarni P. On selecting the right optimizations for virtual machine migration. In: Proc. of the 12th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. ACM, 2016. 37–49.

[34] Elghamrawy K, Franklin D, Chong FT. Predicting memory page stability and its application to memory deduplication and live migration. In: Proc. of the 2017 IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS). IEEE, 2017. 125−126.

[35] Deshpande U, Schlinker B, Adler E, *et al*. Gang migration of virtual machines using cluster-wide deduplication. In: Proc. of the 2013 13th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2013. 394−401.

附中文参考文献:

[18] 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]

陆志刚(1979－),男,江苏苏州人,博士生,高级工程师,主要研究领域为网络分布式计算,软件工程.

徐继伟(1985－),男,博士,助理研究员,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.