计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(4):86−93 [doi: 10.15888/j.cnki.csa.009002]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

容器热迁移优化技术研究①

王智慧, 周忠君

(复旦大学 软件学院, 上海 200433)
通信作者: 王智慧, E-mail: 19212010028@fudan.edu.cn

摘　要: 容器虚拟化技术由于轻量级的特性逐渐在云计算中崭露头角. 容器热迁移是许多云管理能力的基础, 其在最短的宕机时间内, 将运行中的容器完整地迁移到另一个物理节点上继续运行. 性能是容器热迁移研究的重点, 但通过对现有容器热迁移系统的详细分析, 本文发现其中仍然存在着一些影响性能的问题, 包括转储并行度低、预拷贝策略不收敛以及根文件系统与运行状态迁移并行度低等. 针对这些问题, 本文分别提出和设计了资源感知的并行转储机制、基于后拷贝策略的运行状态迁移和基于多优先级的传输调度并行算法等优化策略和算法, 并基于 Docker 实现了一个高性能容器热迁移系统 Dmigrate. 实验结果表明 Dmigrate 相比于目前最新的研究, 平均可有效减少 17.05% 的宕机时间, 总迁移时间平均减少 24.33%.

关键词: 云计算; 虚拟化; 容器; 热迁移; Docker

引用格式: 王智慧,周忠君.容器热迁移优化技术研究.计算机系统应用,2023,32(4):86–93. http://www.c-s-a.org.cn/1003-3254/9002.html

Research on Optimization Technology for Container Live Migration

WANG Zhi-Hui, ZHOU Zhong-Jun

(Software School, Fudan University, Shanghai 200433, China)

Abstract: Container virtualization is emerging in cloud computing due to its lightweight feature. Container live migration is the basis for many cloud management capabilities, which migrates a running container to another physical node with minimal downtime. Performance is the focus of container live migration research, but through a detailed analysis of existing container live migration systems, this study finds that there are still some problems affecting the performance, including low parallelism of dump, non-convergence of pre-copy policy, and low parallelism of root file system and running state migration. To solve these problems, this study proposes and designs three optimization strategies or algorithms including the resource awareness-based parallel dump mechanism, post-copy policy-based running state migration, and multi-priority-based parallel transfer scheduling algorithm. In addition, the paper realizes a high-performance container live migration system, namely, Dmigrate, based on Docker. Experimental results show that compared with the latest research, Dmigrate can effectively reduce downtime by 17.05%, and the total migration time is decreased by 24.33% on average.

Key words: cloud computing; virtualization; container; live migration; Docker

1　引言

云计算在经历了十几年的发展后, 如今已成为信息化发展的重要基础设施[1]. 支撑云计算的关键技术之一是虚拟化技术[2], 目前两种主流的虚拟化形式是虚拟机和容器. 其中容器采用 Namespace[3] 和 Cgroups[4] 机制来隔离和控制物理资源, 并共享宿主机的内核, 与虚拟机相比, 其具有性能更好、资源利用率更高、启动速度更快等优点, 因此受到了越来越多的云计算厂商

① 收稿时间: 2022-08-15; 修改时间: 2022-09-15; 采用时间: 2022-09-27; csa 在线出版时间: 2022-12-23
CNKI 网络首发时间: 2022-12-27

的青睐, 如 Amazon elastic container service[5], IBM bluemix container service[6], Google Kubernetes engine[7] 都是以容器为基础对外提供云计算服务. 但容器技术从根本上来说不同于虚拟机技术, 因此许多云管理技术需要重新基于容器设计和实现, 其中热迁移便是一项基础且重要的云管理技术, 其是容错[8]、服务器合并[9]、负载均衡[10] 等众多云管理能力的基础.

容器热迁移需要迁移容器的运行状态、根文件系统和管理上下文. 运行状态和根文件系统是容器运行所必须的, 运行状态包括内存页和内核上下文, 其中内存页的数据量占比最大. 根文件系统中则保存了容器可能访问的所有文件, 其本质上是宿主机上的一个目录. 管理上下文即容器的各种配置信息, 由容器引擎保存和维护. 性能是容器热迁移研究的重点. 评估一个容器热迁移系统性能的主要指标是宕机时间和总迁移时间. 宕机时间是热迁移过程中容器停止运行的时间, 其长短决定迁移过程对用户体验的影响程度. 总迁移时间是从迁移过程被启动到迁移完成所花费的总时间, 其影响容器重定位的速度.

目前学术界和工业界都存在一些关于容器热迁移的研究. 基于 Docker[11] 的容器转储/恢复功能可以实现容器的热迁移, 但其没有采用任何优化策略, 因此宕机时间较长. LMM[12], P.Haul[13], Sledge[14], CloudHopper[15] 等采用预拷贝策略[16] 来优化容器运行状态和根文件系统的迁移. 所谓的预拷贝策略是一种多次迭代的策略, 它首先在不停止容器运行的情况下将其所有的内存页和根文件系统文件传输到目的端; 然后迭代传输上一次迭代过程中产生的脏数据, 直到脏数据量足够小或达到其他迭代终止条件, 再停止容器的运行, 迁移剩余的容器数据; 最后在目的端重新恢复容器的运行. 预拷贝策略通过多次迭代减少了宕机期间需要迁移的数据量, 缩短了容器热迁移的宕机时间. 但其可能会出现不收敛现象[17], 即在多次迭代后脏数据量不减少, 无法达到停机迁移的阈值, 导致不能有效缩短宕机时间, 甚至迁移失败. 出现这一现象的原因是容器的负载较高时, 其脏数据率较大, 超过了传输速率, 脏数据量就会不减反增. LIMOCE[18]、罗成等[19]、Shang 等[20]、Yu 等[21] 的研究中提出的优化策略提高了内存页的传输速率, 可以在一定程度上缓解预拷贝策略的不收敛问题, 但都不能完全避免. Voyager[22] 利用网络文件系统和联合挂载文件系统设计了容器根文件系统的惰性迁移方案, 所谓的惰性迁移, 是指在容器宕机期间不迁移根文件系统, 而是容器恢复运行之后再迁移. 这种惰性迁移的方式可以大大减少宕机期间的传输数据量, 有效缩短宕机时间, 不存在不收敛问题.

目前, 容器热迁移过程中仍然存在一些影响性能的问题, 导致不能满足一些场景, 如高可用场景, 对高性能容器热迁移系统的需求. 本文针对发现的性能问题, 提出和设计了一系列优化策略.

本文的主要贡献如下.

(1) 针对转储并行度低、预拷贝策略不收敛问题, 提出了并行转储机制和基于后拷贝策略的运行状态迁移来优化容器运行状态的迁移过程.

(2) 设计了基于多优先级的传输调度并行算法来优化根文件系统的迁移过程, 以提高根文件系统迁移与运行状态迁移的并行度, 并调度文件传输顺序, 减少对容器运行性能的影响.

(3) 基于 Docker 实现了一个高性能容器热迁移系统 Dmigrate. 在 Dmigrate 上的性能测试实验显示, 与目前最新的研究相比, 其宕机时间平均减少 17.05%, 总迁移时间平均减少 24.33%.

2 运行状态迁移优化

本节针对运行状态迁移过程中存在的转储并行度低、预拷贝策略不收敛的问题分别设计了资源感知的并行转储机制和基于后拷贝策略的运行状态迁移.

2.1 资源感知的并行转储机制

运行状态的迁移由转储、传输和恢复 3 个过程组成, 转储是收集容器的运行状态并将其保存到文件中的过程. 目前业内公认的转储工具是 CRIU[23], 其借助 /proc 文件系统和 ptrace 机制来收集容器的运行状态. 但其转储过程是完全串行的, 进程之间的转储过程是串行进行的, 每个进程内各种运行状态的转储也是串行进行的. 单个进程的转储过程可抽象为对如图 1 所示的树结构的层次遍历过程. 可以看出, 虽然一些转储过程之间没有依赖关系, 但其仍然是串行进行的. 因此, 现有转储机制的并行度仍有很大的提升空间.

转储并行度的增大可以提高转储的效率, 但同时也意味着更多的资源消耗. 这会与主机上运行的其他程序产生资源竞争, 对它们的运行性能产生影响. 另一方面, 当主机上没有充足的空闲资源时, 转储效率就不会随着转储并行度的增大而提高, 反而会带来额外的

并行开销.

资源感知的并行转储机制的设计目标是最大限度地提高容器运行状态转储过程的并行度, 缩短宕机时间, 同时也要避免占用过多的主机资源, 以免影响主机上其他程序的运行.

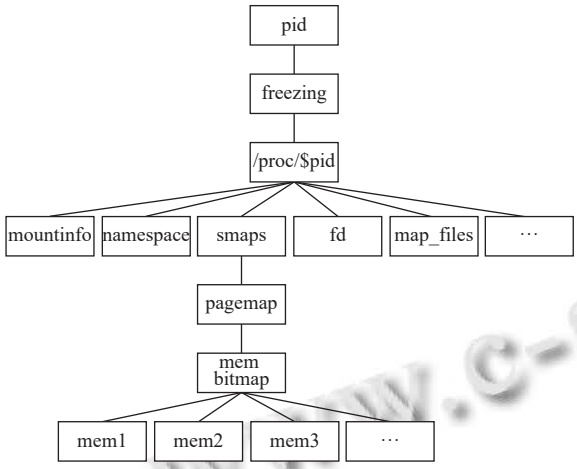

图 1 单个进程的转储过程

如图 2 所示, 资源感知的并行转储机制由资源感知模块、任务列表和线程池 3 部分组成.

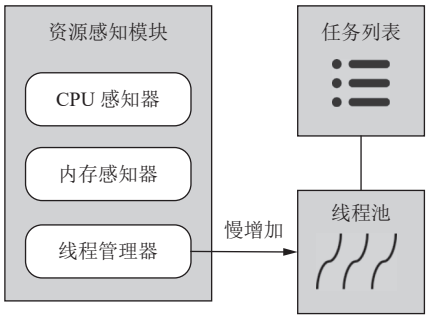

图 2 资源感知的并行转储机制

资源感知模块感知主机上 CPU 和内存的使用情况, 然后据此调整线程池中的线程数量. 其采用慢增加的方式新建线程, 即在开始时只创建一个线程, 线程创建后, 如果感知到主机上还有剩余资源可利用, 则再新创建一个线程, 依次类推, 直到主机上没有充足的资源可以使用, 或者达到最大线程数, 即主机的 CPU 个数.

任务列表中存放待完成的转储任务. 初始时其中只包含一个主任务, 之后主任务将进程树中每个进程的转储控制任务加入其中, 转储控制任务控制单个进程的转储进度, 并将可以并行进行的转储任务加入任务列表中.

线程池中的工作线程从任务列表中获取并执行任务. 工作线程完成当前任务后不马上退出, 而是继续下一个任务. 在完成所有的任务后, 再由资源感知模块的线程管理器销毁所有线程.

2.2 基于后拷贝策略的运行状态迁移

目前的容器热迁移系统[12–15,18–21] 多采用预拷贝策略来迁移容器的运行状态. 预拷贝策略虽然在某些情况下可以减少停机期间的迁移数据量, 缩短宕机时间, 但其存在不收敛问题.

不同于预拷贝策略, 后拷贝策略[24,25] 先停止容器的运行, 将除内存页以外的运行状态转储传输到目的端, 然后基于这些运行状态重新恢复容器的运行, 最后再迁移内存页. 其不存在不收敛问题. CRIU[23] 中实现了进程运行状态的后拷贝转储/恢复功能, 本文在 Docker 中集成了这一功能, 实现了容器的后拷贝转储/恢复.

内存页的迁移方式有两种, 一是按需传输, 即在容器访问缺失的内存页时, 将其从源端传输到目的端. 如图 3 所示, 源端在转储容器运行状态时, 将内存页暂放入管道中, 并启动一个页服务器, 其负责接收来自目的端的页请求. 目的端在恢复迁移容器的运行状态之前, 先启动一个名为惰性页面恢复器的守护进程. 恢复运行状态时, 跳过内存页的恢复, 先不将其加载到容器的内存空间中, 而是将缺失的内存页注册到新创建的 userfaultfd 上, 并将此 userfaultfd 通过套接字发送给惰性页面恢复器. Userfaultfd 是 Linux 上一种可以让用户处理缺页异常的机制. 惰性页面恢复器通过监控此文件描述符即可检测到容器的缺页异常, 并通过将从源端请求的内存页注入到容器的内存空间中来解决缺页异常, 从而实现内存页的按需传输.

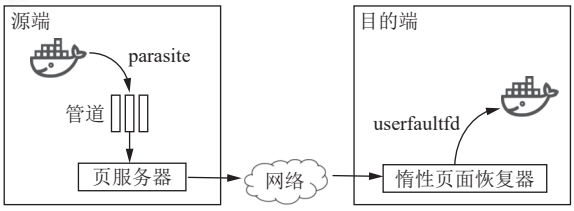

图 3 后拷贝策略中内存页的按需传输

另一种内存页迁移方式是主动传输, 即在容器访问内存页之前就将其从源端传输到目的端. 在按需传输方式下, 发生缺页异常时, 容器要等待内存页从源端通过网络传输到目的端, 因此容器的运行性能会受到一定的影响, 而主动传输方式可以减少按需传输对容器运行性能产生的影响. 在没有缺页异常时, 惰性页面恢复器会主动向源端页服务器请求其他未迁移的内存

页, 将其注入容器的地址空间. 此外, 由于内存访问具有空间局部性[26] 的特点, 源端页服务器会主动传输与缺失内存页相邻的其他内存页, 这样可以减少按需传输内存页的次数, 减少对容器运行性能的影响.

后拷贝策略在宕机期间只需迁移一小部分运行状态, 且这部分运行状态的数据量基本不会随着容器负载的增大而发生很大的变化, 宕机时间也就不会随着负载的增大而增大. 后拷贝策略与预拷贝策略的对比如表 1 所示.

表 1 后拷贝策略与预拷贝策略的对比

对比项	预拷贝策略	后拷贝策略
宕机时间	大	小
总迁移时间	大	小
总网络流量	大	小
迁移时间的可预测性	否	是
性能影响	无	有

3 基于多优先级的传输调度并行算法

Voyager[22] 中设计的根文件系统惰性迁移方案可以大大减少宕机期间的传输数据量, 有效缩短宕机时间, 也不存在不收敛问题, 但容器的运行性能会受到一定的影响, 总迁移时间也存在优化的空间. 本节针对惰性根文件系统迁移方案中存在的问题, 设计了基于多优先级的传输调度并行算法, 其设计目标是提高根文件系统迁移与运行状态迁移之间的并行度, 并调度根文件系统中文件的传输顺序, 以缩短总迁移时间, 降低对容器运行性能的影响.

不同于 Voyager 在运行状态的迁移完成后再迁移根文件系统, 本文在容器被冻结后即开始根文件系统的迁移, 与运行状态的迁移并行进行. 运行状态的转储过程可分为两个阶段——进行冻结阶段和冻结阶段. 两阶段各自所用的时间如表 2 所示, 进行冻结阶段约占总转储时间的 15%, 冻结阶段占总时间的 85% 左右. 进行冻结阶段对容器进行冻结操作, 容器可能还处于运行中, 而冻结阶段容器已经被冻结, 停止运行, 根文件系统也不再变化. 因此在进行冻结阶段之后即可开始根文件系统的迁移, 且其与运行状态的迁移之间有很大的并行空间. 为了获知容器被冻结的准确时间点, 本文在现有转储机制的 action scripts 功能中添加了 POST-FREEZE 阶段, 当容器完成冻结时, 调用用户脚本通知用户.

表 2 转储过程两阶段所用时间

容器	冻结阶段时间 (ms)	进行冻结阶段时间 (ms)	冻结阶段时间/总时间 (%)
Ubuntu	460.60	100.66	82.07
postgres	673.73	101.78	86.88
redis	478.58	0.94	99.80
node	469.35	1.137	99.76
busybox	460.39	100.982	82.01
MySQL	742.097	103.93	87.72
memcached	583.99	101.73	85.16
rabbitmq	654.69	104.42	86.24
httpd	619.00	109.55	84.96
traefik	494.07	1.543	99.69

基于多优先级的传输调度并行算法根据根文件系统中文件的优先级调度其传输顺序. 本节选出了 3 类高优先级文件, 即 3 类被访问可能性较高的文件.

(1) 工作文件集

为了解答哪些文件在容器迁移之后被访问的可能性较大这一问题, 本节设计了一个预实验. 预实验选取了 Docker Hub[27] 上下载量最高的 24 个容器镜像, 基于这些镜像运行容器, 统计容器迁移之后访问的前 10 个文件. 实验结果如表 3 所示, 容器迁移之后访问的前 10 个文件中 48.33% 为程序二进制文件、已经打开的文件和映射文件.

表 3 容器迁移后访问的前 10 个文件

文件类别	访问次数	占比 (%)
程序二进制文件	27	11.25
打开的文件	48	20
映射文件	41	17.08
其他文件	124	51.67

基于预实验的结果, 本文提出了工作文件集的概念, 即容器迁移时其中运行的程序的二进制文件、已打开的文件和映射文件的集合, 其可分别通过/proc/$pid/fd, /proc/pid/mapfiles,/proc/pid/mapfiles,/proc/pid/exe 获取. 工作文件集中的文件在容器迁移之后的短时间内很可能会被访问, 在文件传输中具有最高的优先级.

(2) 同目录文件

应用程序倾向于在一段时间内访问相互关联的数据, 而在实践中, 用户或程序通常把相互有关联的文件放在同一个目录下, 因此, 在容器访问过一个文件之后, 接下来同目录下的文件相比于其他文件被访问的可能性更大. 我们通过 inotify 机制监控迁移容器的文件操作, 以此获取同目录文件. 同目录文件在文件传输中具有第二优先级.

(3) 小文件

一个文件是一个整体, 只有文件中的所有数据都从源端传输到目的端, 迁移容器才能从本地访问这个文件. 优先传输小文件可以在短时间内完成更多文件的传输, 从而提高迁移容器对本地文件的命中率. 假设有一组文件 f1、f2、f3、f4, 其大小分别为 32 MB、64 MB、128 MB、256 MB, 源端与目的端之间的网络传输速率为 512 MB/s, 则每个文件所需传输时间分别为 62.5 ms、125 ms、250 ms、500 ms. 另假设迁移容器在恢复运行之后每隔 100 ms 随机访问一个文件, 前 1 s 内的文件命中率期望值为 E_{HR}. 如果以随机方式传输文件, 则每次的本地文件命中概率如表 4 第 1 列所示, 命中率期望值为:

$$E_{HR}=\frac{\frac{1}{16}+\frac{1}{6}+\frac{11}{48}+\frac{5}{16}+\frac{7}{16}+\frac{15}{32}+\frac{7}{12}+\frac{5}{8}+\frac{3}{4}+1}{10}\approx 0.46 \tag{1}$$

表 4 随机传输文件和优先传输小文件时本地文件命中概率

次数	随机传输文件	优先传输小文件
1	1/16	1/4
2	1/6	1/2
3	11/48	1/2
4	5/16	1/2
5	7/16	3/4
6	15/32	3/4
7	7/12	3/4
8	5/8	3/4
9	3/4	3/4
10	1	1

如果优先传输小文件, 即按照 f1、f2、f3、f4 的顺序传输文件, 每次的本地文件命中概率如表 4 第 2 列所示, 命中率期望值为:

$$E_{HR}=\frac{\frac{1}{4}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{3}{4}+\frac{3}{4}+\frac{3}{4}+\frac{3}{4}+\frac{3}{4}+1}{10}=0.65 \tag{2}$$

相比于随机传输文件, 命中率提高了大约 19%. 因此优先传输小文件可以提高容器对本地文件的命中率.

基于多优先级的传输调度并行算法在传输文件时首先设置一个已传输文件列表和 3 个优先级栈, 第一优先级栈存放工作文件集文件, 第二优先级栈存放同目录文件, 第三优先级栈存放其他文件. 第一和第三优先级栈中的文件按照文件大小排序, 小文件在栈顶, 优先被传输. 第二优先级栈初始时为空, 在容器恢复运行之后, 如果监测到其对某一文件的读写操作, 则将该文件同目录下的其他未迁移文件按照文件大小从大到小的顺序放入第二优先级栈. 在传输的过程中, 先传输第一优先级栈内的文件, 只有第一优先级栈为空时, 才开始传输第二优先级栈内的文件, 依次类推. 传输过程是非抢占的, 即如果在传输第三优先级栈内的文件时, 第二优先级栈内有新文件到达, 则在完成当前文件传输之后, 再传输第二优先级栈内的文件.

4 实验分析

我们在两台 Ubuntu 16.04 LTS 虚拟机上对本文设计与实现的容器热迁移系统——Dmigrate 进行了性能测试实验, 其中一台作为迁移源端, 另一台作为迁移目的端. 每台虚拟机配置有 4 个 vCPU, 64 GB 内存, 100 GB 磁盘存储和 ext4 文件系统. 两台虚拟机之间的平均网络带宽为 1 Gb/s.

我们选取了 Docker Hub[27] 上下载量最高的 24 个容器镜像进行了实验, 从宕机时间和总迁移时间两个方面来评估 Dmigrate 的性能优化程度. 所有的实验均进行 10 次, 取其平均值作为最终的结果.

4.1 宕机时间

宕机时间是评估一个容器热迁移系统性能的最重要指标. 本节首先测量了 Dmigrate 和其他相关工作的宕机时间, 迁移工作负载包括数据库类工作负载 (MySQL)、操作系统类工作负载 (Ubuntu)、语言环境类工作负载 (Python)、Web 服务类工作负载 (wordpress)、中间件类工作负载 (rabbitmq). 为了表述方便, 后文将 Ma 等[12] 的方法简称为 LMM, Yu 等[21] 的方法简称为 LMY, Shang 等[20] 的方法简称为 LMS, 罗成等[19] 的方法简称为 LML. 实验结果如图 4 所示. 其中基于 Docker 转储/恢复功能的迁移方法宕机时间最长, 其未采用任何优化策略. 与 Docker 迁移方法相比, Voyager[22] 的宕机时间有所下降, 其采用了根文件系统惰性迁移方法, 在宕机期间无需迁移根文件系统, 只需迁移运行状态, 宕机期间的迁移数据量大大减少. LMM[12]、P.Haul[13]、Sledge[14]、CloudHopper[15] 的宕机时间相近, 且相比于 Docker 迁移方法都有所下降, 因为其都在正式迁移前进行一次或多次预拷贝, 在宕机期间只需迁移脏数据. LIMOCE[18]、LML[19]、LMS[20]、LMY[21] 也都采用预拷贝策略, 并在此基础上提出了其他的优化策略, 实验结果表明, LMY 中的优化策略可以进一步缩短宕机时间, 而 LIMOCE、LML、LMS 中的优化策略在本文的实验环境下对宕机时间几乎没有影响.

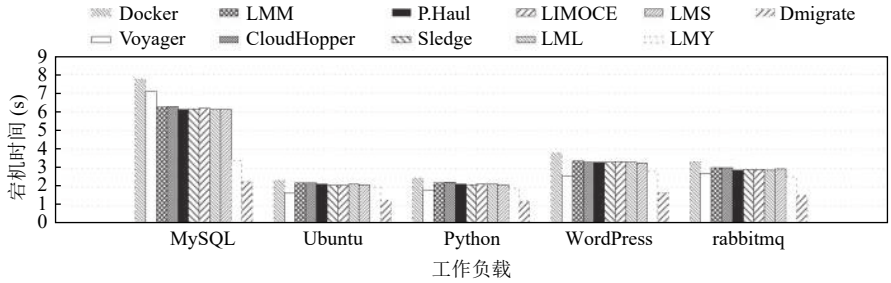

图 4 不同迁移方法宕机时间的对比

为了详细评估本文提出的各项优化策略对宕机时间的优化程度, 本节采用上述方法中所有可以有效缩短宕机时间的优化策略作为基准组, 包括 Sledge[14] 的预拷贝策略、Yu 等[21] 的内存文件系统、Voyager[22] 的惰性迁移方案, 分别测量了本文提出的优化策略对宕机时间的优化程度. 实验结果如图 5 所示, 基准组的宕机时间标准化为 1, 后拷贝策略平均可减少 12.91% 的宕机时间, 资源感知的并行转储机制平均减少 4.14% 的宕机时间, 即 Dmigrate 相比于目前最新的研究可平均缩短 17.05% 的宕机时间. 基于后拷贝策略的运行状态迁移在宕机期间无需转储和恢复内存页, 转储和恢复的数据量减少, 转储和恢复时间都随之缩短. 而资源感知的并行转储机制提高了转储过程的并行度, 也可以有效减少转储时间. 基于多优先级的传输调度并行算法对宕机过程没有影响, 资源感知的并行转储机制和基于后拷贝策略的运行状态迁移都可以有效缩短宕机时间.

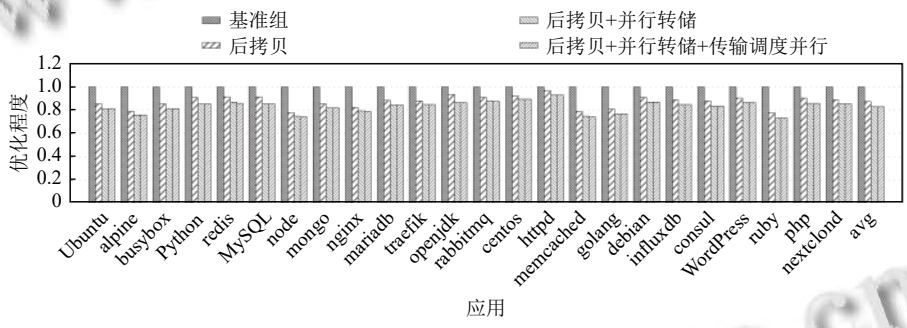

图 5 不同优化策略对宕机时间的优化程度

为了进一步评估 Dmigrate 对宕机时间的优化程度, 我们使用 YCSB[28] 模拟了 redis 和 MySQL 容器的不同负载, 测量了其在不同吞吐量下的宕机时间, 实验结果如图 6、图 7 所示. 从图中可以看出, 基准组的宕机时间随着吞吐量的增大有增大的趋势, 而 Dmigrate 的宕机时间基本保持稳定. Dmigrate 所采用的后拷贝策略在宕机期间不迁移容器的内存页, 也不迁移根文件系统, 所以其宕机期间所迁移的数据量不会随着吞吐量的增大而发生较大的变化, 宕机时间也不会随着吞吐量的增大而增大.

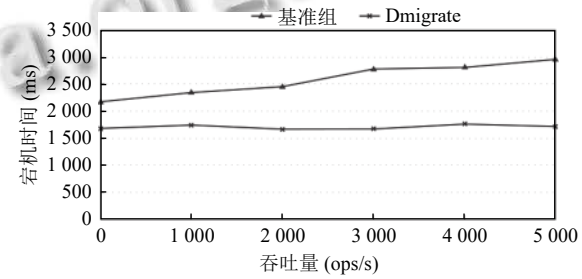

图 6 Redis 容器在不同吞吐量下的宕机时间

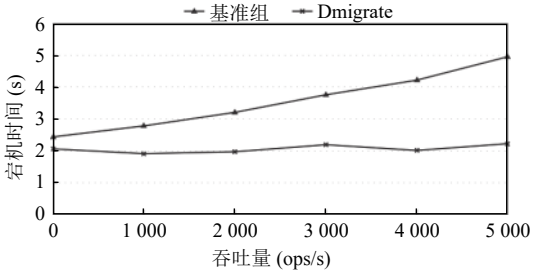

图 7 MySQL 容器在不同吞吐量下的宕机时间

4.2 总迁移时间

总迁移时间是评估容器热迁移系统性能的重要指标之一, 其也反映了惰性迁移对容器运行性能的影响程度, 总迁移时间越短, 对容器运行性能的影响程度就越低.

本节首先测量了 Dmigrate 和其他相关工作的总迁移时间, 实验结果如图 8 所示. Voyager[22] 与基于 Docker 转储/恢复功能的迁移方法总迁移时间没有明显差别, 其在迁移过程中全部数据都只需迁移一次. LMM[12]、P.Haul[13]、Sledge[14]、CloudHopper[15] 的总迁移时间相比于 Voyager 和 Docker 迁移方法都有所增加, 因为其需要预拷贝一次或多次内存页和根文件系统, 预拷贝过程和正式迁移过程之间有重复数据, 导致迁移的总数据量增加, 总迁移时间也增加. LIMOCE[18]、LML[19]、LMS[20]、LMY[21] 也都采用多次迭代的预拷贝策略, 并在此基础上提出了其他的优化策略. LMY采用内存文件系统保存转储文件, 减少了迁移过程中读磁盘文件和写磁盘文件的开销. LMS 推迟迁移访问频率较高的内存页, 可以减内存页的重复迁移. LIMOCE 和 LML 采用的压缩方法在本文的实验环境下对总迁移时间没有明显影响.

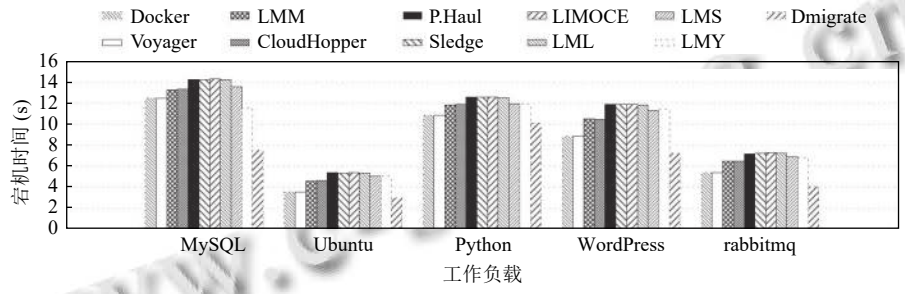

图 8 不同迁移方法总迁移时间的对比

为了详细评估本文提出的各项优化策略对总迁移时间的优化程度, 本节采用上述方法中总迁移时间最短的 Voyager[22] 作为基准组, 分别测量了本文提出的优化策略对总迁移时间的优化程度. 实验结果如图 9 所示, 后拷贝策略平均减少 9.01% 的总迁移时间, 资源感知的并行转储机制平均减少 1.77% 的总迁移时间, 基于多优先级的传输调度并行算法平均减少 13.55% 的总迁移时间, 即 Dmigrate 可平均减少 24.33% 的总迁移时间. 使用后拷贝策略后内存页的迁移可以在后迁移阶段与根文件系统的迁移并行进行, 资源感知的并行转储机制提高了转储过程的并行度, 基于多优先级的传输调度并行算法并行化根文件系统迁移与非内存页运行状态迁移过程. Dmigrate 中所有的数据都只需迁移一次, 且整个迁移过程的并行度与基准组相比大大提高.

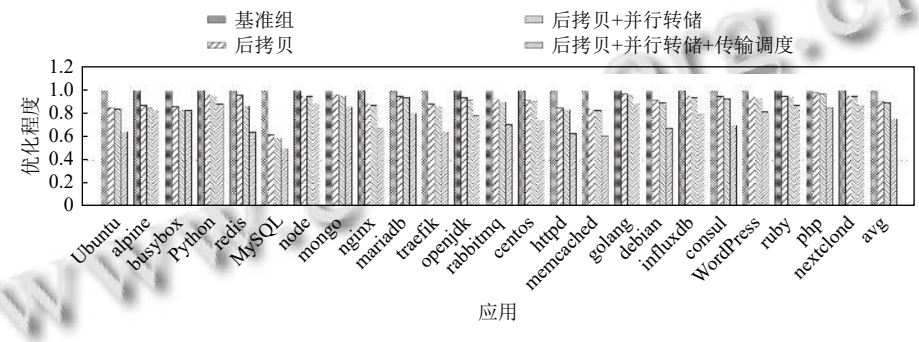

图 9 不同优化策略对总迁移时间的优化程度

5 结论与展望

本文对目前容器热迁移系统中存在的性能问题进行了详细分析, 并针对这些问题提出和设计了一系列优化策略, 实现了一个高性能容器热迁移系统 Dmigrate. 首先, 针对转储并行度低的问题, 设计了资源感知的并行转储机制, 提高转储过程的并行度, 缩短转储时间, 进而减少宕机时间. 其次, 为了解决基于预拷贝策略的运行状态迁移中存在的不收敛问题, 设计了基于后拷贝策略的运行状态迁移, 在容器恢复运行之后再迁移内存页, 不会存在脏内存页不收敛问题. 最后, 为了减少根文件系统惰性迁移对容器性能的影响, 提出了基于多优先级的传输调度并行算法, 根文件系统传输与运行状态迁移并行进行, 并调度文件传输过程以优先传输被访问可能性较大的文件. 实验结果表明 Dmigrate 可以有效减少 17.05% 的宕机时间, 24.33% 的总迁移时间.

未来的工作可以继续探讨容器热迁移的性能进一

步优化的可能. 接下来可以继续探索迁移过程并行度提升的可能性, 或者充分利用容器和根文件系统本身的特点来做优化, 或将高性能网络 RDMA 应用于容器的热迁移过程中. 另外, 未来会将容器的热迁移集成到容器编排工具, 如目前最流行的 Kubernetes 中. Kubernetes 中的编排单元是 pod, 一个 pod 由一个或多个容器组成, 集成工作需要将容器的热迁移扩展为 pod 的热迁移.

参考文献

1 云原生产业联盟. 云原生发展白皮书. 云原生产业联盟, 2020.

2 Rashid A, Chaturvedi A. Virtualization and its role in cloud computing environment. International Journal of Computer Sciences and Engineering, 2019, 7(4): 1131–1136. [doi: 10.26438/ijcse/v7i4.11311136]

3 Namespace in operation. https://lwn.net/Articles/531114/. [2022-08-14].

4 Introduction to control groups. https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/7/html/resource_management_guide/chap-introduction_to_control_groups. [2022-08-14].

5 Amazon. EC2 container service. https://aws.amazon.com/ecs/. [2022-08-14].

6 IBM Inc. IBM cloud kubernetes service. https://www.ibm.com/cloud/kubernetes-service. [2022-08-14].

7 Google Inc. Kubernetes engine. https://cloud.google.com/kubernetes-engine. [2022-08-14].

8 Kumari P, Kaur P. A survey of fault tolerance in cloud computing. Journal of King Saud University—Computer and Information Sciences, 2021, 33(10): 1159–1176. [doi: 10.1016/j.jksuci.2018.09.021]

9 Chaurasia N, Kumar M, Chaudhry R, *et al*. Comprehensive survey on energy-aware server consolidation techniques in cloud computing. The Journal of Supercomputing, 2021, 77(10): 11682–11737. [doi: 10.1007/s11227-021-03760-1]

10 Mishra SK, Sahoo B, Parida PP. Load balancing in cloud computing: A big picture. Journal of King Saud University—Computer and Information Sciences, 2020, 32(2): 149–158. [doi: 10.1016/j.jksuci.2018.01.003]

11 Anderson C. Docker [Software engineering]. IEEE Software, 2015, 32(3): 102–c3. [doi: 10.1109/MS.2015.62]

12 Ma LL, Yi SH, Li Q. Efficient service handoff across edge servers via docker container migration. Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing. San Jose: ACM, 2017. 11.

13 P.Haul. Container live migration. https://criu.org/P.Haul. (2019-10-31).

14 Xu B, Wu S, Xiao J, *et al*. Sledge: Towards efficient live migration of docker containers. Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD). Beijing: IEEE, 2020. 321–328.

15 Benjaponpitak T, Karakate M, Sripanidkulchai K. Enabling live migration of containerized applications across clouds. Proceedings of the 2020 IEEE Conference on Computer Communications. Toronto: IEEE, 2020. 2529–2538.

16 Clark C, Fraser K, Hand S, *et al*. Live migration of virtual machines. Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation. Boston, 2005. 273–286.

17 Zhang F, Liu GM, Fu XM, *et al*. A survey on virtual machine migration: Challenges, techniques, and open issues. IEEE Communications Surveys & Tutorials, 2018, 20(2): 1206–1243.

18 Das R, Sidhanta S. LIMOCE: Live migration of containers in the edge. Proceedings of 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). Melbourne: IEEE, 2021. 606–609.

19 罗成, 崔勇, 林予松. 基于带宽预测与自适应压缩的容器迁移方法. 计算机工程, 2022, 48(5): 200–207, 214.

20 Shang YS, Lv DF, Liu JL, *et al*. Container memory live migration in wide area network. Proceedings of the 5th International Conference on Smart Computing and Communication. Paris: Springer, 2020. 68–78.

21 Yu ZX, He KJ, Chen C, *et al*. Live container migration via pre-restore and random access memory. Proceedings of the 2020 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). Exeter: IEEE, 2020. 102–109.

22 Nadgowda S, Suneja S, Bila N, *et al*. Voyager: Complete container state migration. Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Atlanta: IEEE, 2017. 2137–2142.

23 CRIU Community. Checkpoint/restart in userspace (CRIU). https://criu.org/. [2022-08-14].

24 Hines MR, Deshpande U, Gopalan K. Post-copy live migration of virtual machines. ACM SIGOPS Operating Systems Review, 2009, 43(3): 14–26. [doi: 10.1145/1618525.1618528]

25 Hirofuchi T, Nakada H, Itoh S, *et al*. Enabling instantaneous relocation of virtual machines with a lightweight vmm extension. Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. Melbourne: IEEE, 2010. 73–83.

26 Denning PJ. The locality principle. Communications of the ACM, 2005, 48(7): 19–24. [doi: 10.1145/1070838.1070856]

27 DockerHub. https://registry.hub.Docker.com. [2022-08-14].

28 Cooper BF, Silberstein A, Tam E, *et al*. Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM Symposium on Cloud Computing. Indianapolis: ACM, 2010. 143–154.

(校对责编: 孙君艳)