软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(12):5527−5551 [doi: 10.13328/j.cnki.jos.006765]

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

从 Docker 容器看容器技术的发展: 一种系统文献综述的视角*

吴逸文[1,2,3], 张 洋[1,2,3], 王 涛[1,2,3], 王怀民[1,2,3]

[1](国防科技大学 并行与分布处理国防科技重点实验室, 湖南 长沙 410073)

[2](国防科技大学 计算机学院, 湖南 长沙 410073)

[3](湖南省复杂系统软件工程重点实验室, 湖南 长沙 410073)

通信作者: 张洋, E-mail: yangzhang15@nudt.edu.cn

摘 要: 近些年, 软件构造、运行和演化过程面临着诸多新需求, 例如开发测试环境需要高效切换或配置、应用隔离、减少资源消耗、提高测试和部署效率等, 给开发人员开发和维护软件带来了巨大的负担. 容器技术有希望将开发人员从繁重的开发运维负担中解脱出来, 尤其是 Docker 作为目前工业界的容器行业标准, 近年来逐渐成为学术界一个热门的研究领域. 为了帮助研究人员全面准确地理解当前 Docker 容器研究的现状和趋势, 使用系统文献综述 (systematic literature review) 的方法搜集了 75 篇该领域最新的高水平论文, 进行了详细的分析和总结. 首先, 使用定量研究方法调查了 Docker 容器研究的基本现状, 包括研究数量、研究质量、研究领域和研究方式. 其次, 首次提出了面向 Docker 容器研究的分类框架, 分别从核心、平台和支持 3 个方面对当前研究进行了系统性地归纳和梳理. 最后, 讨论了 Docker 容器技术的发展趋势并总结了 7 个未来的研究方向.

关键词: 容器; Docker; 系统文献综述

中图法分类号: TP311

中文引用格式: 吴逸文, 张洋, 王涛, 王怀民. 从Docker容器看容器技术的发展: 一种系统文献综述的视角. 软件学报, 2023, 34(12): 5527–5551. http://www.jos.org.cn/1000-9825/6765.htm

英文引用格式: Wu YW, Zhang Y, Wang T, Wang HM. Development Exploration of Container Technology Through Docker Containers: A Systematic Literature Review Perspective. Ruan Jian Xue Bao/Journal of Software, 2023, 34(12): 5527–5551 (in Chinese). http://www.jos.org.cn/1000-9825/6765.htm

Development Exploration of Container Technology Through Docker Containers: A Systematic Literature Review Perspective

WU Yi-Wen[1,2,3], ZHANG Yang[1,2,3], WANG Tao[1,2,3], WANG Huai-Min[1,2,3]

[1](Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China)

[2](College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

[3](Laboratory of Software Engineering for Complex Systems, Changsha 410073, China)

Abstract: In recent years, software construction, operation, and evolution have encountered many new requirements, such as the need for efficient switching or configuration in development and testing environments, application isolation, resource consumption reduction, and higher efficiency of testing and deployment. These requirements pose great challenges to developers in developing and maintaining software. Container technology has the potential of releasing developers from the heavy workload of development and maintenance. Of particular note, Docker, as the de facto industrial standard for containers, has recently become a popular research area in the academic community. To help researchers understand the status and trends of research on Docker containers, this study conducts a systematic literature review by collecting 75 high-level papers in this field. First, quantitative methods are used to investigate the basic status of

* 基金项目: 科技创新 2030—“新一代人工智能”重大项目 (2021ZD0112904); 国防科技重点实验室基金 (2021-KJWPDL-06)
收稿时间: 2022-05-18; 修改时间: 2022-06-30; 采用时间: 2022-08-19; jos 在线出版时间: 2023-02-15
CNKI 网络首发时间: 2023-02-16

research on Docker containers, including research quantity, research quality, research areas, and research methods. Second, the first classification framework for research on Docker containers is presented in this study, and the current studies are systematically classified and reviewed from the dimensions of the core, platform, and support. Finally, the development trends of Docker container technology are discussed, and seven future research directions are summarized.

Key words: container; Docker; systematic literature review

1 引 言

近年来, 软件构造、运行和演化过程面临着诸多变化. 传统软件的集中式架构难以满足应用需求, 逐渐演变为分布式架构, 又进一步发展为流行的微服务架构. 同时, 软件的运行环境也从传统的物理机发展到虚拟机, 近些年更是趋向于在云端运行. 这些软件在架构设计、开发方式、部署维护等各个阶段都面临着全新的需求, 例如开发测试环境需要高效切换或配置、应用隔离、减少资源消耗、提高测试和部署效率等, 从而给开发人员开发和维护软件带来了巨大的负担.

容器 (container) 技术作为一种轻量级的操作系统层面的虚拟化技术, 能够为软件应用及其依赖组件提供一个资源独立的运行环境[1]. 在容器化过程中, 应用程序及其所有必要的依赖关系 (代码、运行时、系统工具和系统库等) 会被打包成一个可重用的镜像 (image), 镜像内容可以通过配置文件 (例如 Dockerfile 和 docker-compose.yml) 中的指令定义[2], 镜像运行环境不与主操作系统共享内存、CPU 和硬盘空间, 由此也保证了容器内部进程与容器外部进程的独立关系[3]. 与传统虚拟机相比, 容器技术具有快速启动、运行环境可移植、弹性伸缩、快速部署以及低系统资源消耗等优点[4–7]. 随着云原生[8]、DevOps[9]、CI/CD[10]等愈发流行与普及, 容器技术在软件开发和运维方面有着广阔的应用前景[10,11], 已经引起了业内的广泛关注[12]. 许多云服务提供商已经开始提供基于容器的云服务, 以满足其不断增长的需求, 例如谷歌容器引擎 (Google GKE)、亚马逊弹性容器服务 (Amazon ECS) 和 Azure 容器服务 (Azure ACS).

在众多容器技术中, Docker 无疑是最受欢迎的容器化技术, 已成为工业界的行业标准[12,13]. Docker 本身是一个基于 LXC (Linux 容器)[14]、操作系统虚拟化[2]等技术实现的开源项目, 通过提供标准化的运行时环境, 可以将同一个软件构建版本用于开发、测试、预发布、生产等任何环境, 并且与底层操作系统解耦, 从而实现“构建一次, 随处运行”的目标. Docker 容器技术有希望将开发人员从繁重的开发运维负担中解脱出来, 其主要原因有 3 个: (1) Docker 可以很好地解决代码运行环境变更问题, 从而降低依赖时的复杂度; (2) Docker 可以通过定义环境, 很好地解决环境不一致的问题; (3) Docker 可以帮助开发者精简部署流程, 使代码部署更加透明、高效.

自 2013 年诞生至今, Docker 容器镜像已经被下载超过 1 300 亿次 (https://www.docker.com/company). 《Flexera 2021 年云计算报告》[15]发现, 76% 的公司受访者表示正在或计划使用 Docker 容器技术. 目前 Docker 受到越来越多的公司和开发者欢迎, 被广泛地应用于软件开发和运维过程[13], 也在各种实际应用中部署, 例如智能汽车、物联网和雾/边缘计算 (fog/edge computing)[16–19]. 近年来, Docker 容器技术也逐渐成为学术界一个热门的研究领域, 相关论文发表在软件工程领域的高质量会议和期刊上, 如 ICSE、FSE 和 ESE 等. 此外, 在并行与分布计算 (如 TPDS) 以及存储系统 (如 FAST) 等领域的高质量会议上也有相关论文发表.

尽管 Docker 容器技术在工业界已经非常受欢迎, 并且学术界对其也开展了一系列探索研究. 但是, 目前还没有研究工作全面准确地梳理和总结当前 Docker 容器研究的现状和趋势. 为了帮助研究人员及时清楚地掌握这一重要研究领域的相关工作, 本文使用系统文献综述方法对 Docker 容器研究的已有工作进行了全面分析. 首先, 本文从不同文献数据库搜集了 75 篇该领域最新的高水平论文, 并对它们的数量、质量、研究领域和研究方式进行了定量分析. 进一步, 本文首次提出了面向 Docker 容器研究的分类框架, 从核心、平台和支持 3 方面对当前研究进行了系统性地归纳和梳理. 最后, 本文讨论了 Docker 容器技术的发展趋势并总结了未来的研究方向, 以期为相关的研究人员提供参考.

综上所述, 本文的主要贡献总结如下.

(1) Docker 容器是当前国内外较新的一个研究方向, 目前还没有专门针对 Docker 容器的综述文章. 本文是第

一篇系统总结 Docker 容器研究进展的中文综述, 提供了该方向较全面的研究工作总结和展望.

(2) 提出了面向 Docker 容器研究的分类框架, 该框架从核心、平台和支持 3 个方面对当前研究进行了分类和总结, 可以为后续研究提供参考.

(3) 讨论了 Docker 容器技术的发展趋势, 总结了 7 个 Docker 容器未来可能的研究方向. 这些方向基于我们对现有工作的深入思考, 对相关研究人员有一定的参考价值.

本文第 1 节介绍相关研究背景和主要贡献. 第 2 节系统地介绍综述研究过程, 包含规划、实施流程和效度威胁. 第 3 节介绍了当前 Docker 容器研究的基本现状, 包括研究数量、研究质量、研究领域和研究方式. 第 4 节详细阐述了 Docker 容器研究的主题分类结果, 包括核心研究、平台研究和支持研究 3 个类别. 第 5 节讨论了 Docker 容器技术的发展趋势以及未来的研究方向. 第 6 节总结全文.

2 研究方法

本文主要采用系统文献综述方法 (systematic literature review, SLR) 开展研究. SLR 旨在识别、评估和解释有关特定研究问题、主题领域和感兴趣现象的所有相关研究[20]. 如图 1 所示, 本文基于文献 [21] 中的指南, 采用了规划 (plan)、实施 (conduct) 和报告 (report) 这 3 步综述研究过程. 其中, 规划阶段包括定义研究问题、确定搜索范围以及制定文献筛选标准, 实施阶段主要进行文献的检索、选择、分类以及综合分析, 报告阶段则将调研的结果进行系统性展示.

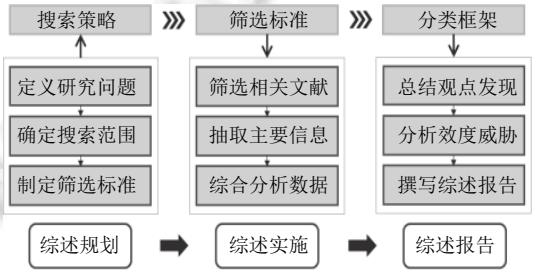


图 1 本文系统文献综述过程

2.1 综述规划

在规划阶段, 本文以“docker”为关键词, 在 IEEE Xplore Digital Library、ACM Digital Library、Springer Digital Library、Elsevier Science Direct 以及 CNKI (知网) 等文献数据库中执行检索操作. 这些数据库是关于计算机科学潜在相关研究的主要文献来源[22]. 特别地, 本文采取了“迭代查找”[23]的论文收集方法: 首先在已搜集论文的参考文献中查找符合标准的新论文, 然后将新发现的论文加入已获取的论文集合中, 重复上述步骤直到不再发现新论文为止. 该方法能够尽可能多地覆盖目前的相关工作. 此外, 制定筛选标准有助于确定与本综述主题直接相关的研究, 保证纳入论文的质量、时效性、相关性和可及性. 本文在搜索过程中执行的筛选标准如下.

- 论文发表时间在 2014–2021 年之间.
- 论文页数是 8 页及以上.
- 论文需经过同行审阅评议.
- 论文可在线获取.
- 排除不属于国内外计算机领域评级的研究论文.
- 排除只是简单提及 Docker 而非对其进行深入研究的研究论文.

2.2 综述实施

在实施阶段, 本文主要执行以下 4 个步骤.

步骤 1. 基于搜索引擎的自动检索. 本文在 5 个数据库中进行关键字检索, 并在自动搜索过程中使用搜索词与

数据库中的全文相匹配[24], 以避免错过任何可能与 Docker 容器有关的潜在论文. 通过将检索领域 (例如 Springer 的“discipline”和 Elsevier 的“subject area”) 设置为计算机科学 (computer science), 本文只关注 Docker 容器在计算机科学领域的相关研究. 此外, 本文设置的检索时间区间为 2014–2021 年, 因为 2013 年 3 月 Docker 创始人 SolomonHykes 在 PyCon 大会上首次公开介绍了 Docker, 在 2014 年 6 月的 DockerCon 大会上 Docker 正式发布了 Docker 1.0 版本. 此后, Docker 的创新镜像格式以及容器运行时迅速成为社区、客户和行业的实际标准和基石. 综合以上事实, 我们认为从 2014 年开始搜索能够较全面地覆盖此领域的现有研究. 这一步骤共检索到 7 875 篇论文 (IEEE: 927 篇; ACM: 3 515 篇; Elsevier: 911 篇; Springer: 1 718 篇; CNKI: 804 篇).

步骤 2. 基于关键信息的人工筛选. 通过检查论文页数并阅读标题、关键词、摘要和结论, 本文对检索到的论文进行了初步筛选, 删除了不足 8 页、非计算机科学领域、只是简单提及 Docker 而非对其深入研究的相关论文 (共 7 481 篇), 当不确定论文是否符合筛选标准时暂时保留以待全文判断. 进一步, 本文通过阅读研究论文的全文, 排除所有不符合筛选标准的研究论文 (共 323 篇). 为了确保论文的质量水平, 本文参考中国计算机学会 (CCF) 和澳大利亚计算机研究和教育协会 (CORE) 推荐的学术期刊和会议列表, 只将符合评级的论文纳入文献综述集合, 此步骤获得 71 篇论文 (IEEE: 30 篇; ACM: 10 篇; Elsevier: 11 篇; Springer: 18 篇; CNKI: 2 篇).

步骤 3. 基于迭代查找的论文补充. 在人工筛选完成后, 为了不遗漏重要论文, 本文采用了“迭代查找”的论文收集方法, 即检查现有论文的参考文献, 从中找出可能相关的论文. 此步骤共找出 23 篇相关论文, 经过人工筛选, 其中有 4 篇论文符合标准. 因此, 最终文献综述集合共有 75 篇论文 (IEEE: 31 篇; ACM: 10 篇; Elsevier: 11 篇; Springer: 19 篇; CNKI: 2 篇; USENIX: 2 篇).

步骤 4. 基于特征维度的数据分析. 如表 1 所述, 本文遵循先前文献综述分析[24]的特征维度, 通过抽取结构化信息获取论文的基础属性特征. 对于论文的研究方式和研究分类特征, 本文由第一作者和第二作者分别阅读每篇论文并执行分类操作, 所参考依据包括相关文献[25]、官方文档[26]、在线博客资料[27–29]等, 其余两位作者参与讨论, 最终分类结果需满足所有作者一致意见. Fleiss’s Kappa 检验结果 (贡献类型: 0.90; 评估方法: 0.87; 研究类别: 0.92; 研究主题: 0.83) 表明本文最终分类结果符合统计学一致性要求[30]. 进一步, 本文对相关数据进行了定量分析, 以描述 Docker 容器研究的基本情况 (见第 3 节).

表 1 数据分析结构化表

特征	指标	描述	取值
基础属性	年份	论文发表时间	2014–2021
	出处	论文发表期刊或会议名称	ICSE/ASE/FSE/软件学报等
	出版社	论文所属出版社	ACM/IEEE/SPRINGER/CNKI等
	发表类型	论文发表类型	期刊/会议
	质量评级	论文发表期刊或会议评级	A(A*)/B/C/无评级
	研究领域	论文所属研究领域	软件工程/计算机体系结构等
研究方式	贡献类型	论文研究主要贡献类型	解决方案/评估研究/哲学/观点/经验
	评估方法	论文研究采用的评估方法	控制实验/案例研究/经验报告等
研究分类	研究类别	论文研究所属的类别	核心研究/平台研究/支持研究
	研究主题	论文研究主要关注的主题	容器配置/容器镜像/资源调度/集群管理/服务部署/安全分析/性能分析

2.3 效度威胁

首先, 本文并没有对所有文献数据库执行检索操作, 因此可能会遗漏其他相关的 Docker 容器研究论文. 为了尽可能地降低威胁, 本文遵循先前相关工作的筛选方法[22], 选择 IEEE、ACM、Springer、Elsevier 以及 CNKI (知网) 这 5 个著名的文献数据库作为文献来源. 同时, 本文采用“迭代查找”的论文收集方法用于补充其他可能相关的论文. 其次, 在检查论文的研究方式和研究分类特征时, 本文主要采用人工分析方法, 可能会受到主观判断误差的影响. 为了更加准确地获得论文主题和特征, 本文两位作者分别阅读每篇论文并执行分类操作, 其余两位作者参与

讨论, 最终分类结果满足了 Fleiss's Kappa 检验的统计学一致性要求. 最后, 本文主要对 75 篇论文进行分析, 因此相关发现和结论可能不适用于所有 Docker 容器研究论文. 为了保证最终综述集合中论文的代表性和质量水平, 本文只考虑属于国内外计算机领域评级 (CCF 评级和 CORE 评级) 的论文.

3 Docker 容器研究现状分析

本节旨在调查 Docker 容器研究的基本现状, 包括研究数量、研究质量、研究领域和研究方式.

3.1 研究数量

文献综述集合共有 75 篇论文, 其中期刊论文 21 篇 (占比 28.0%, 发表在 13 种不同期刊), 会议论文 54 篇 (占比 72.0%, 发表在 33 个不同会议). 结果表明, 一方面众多期刊和会议收录本领域的论文, 论文发表呈现多样化的特点; 另一方面, 还没有显著占优的期刊或国际会议成为本领域研究者发文的首选. 进一步, 本文分析了 Docker 容器研究的论文发表数量随时间的演变情况. 如图 2 所示, 随着时间发展该领域论文发表数量整体上 (除 2021 年) 呈上升趋势, 这表明相关研究近年来受到了越来越多的关注. 同时会议论文的数量是期刊论文的两倍以上, 说明研究人员主要参加会议, 因为他们可以在最短的时间内获得反馈并发表研究成果, 这也表明这个领域正在快速发展. 2014 年 6 月 Docker 1.0 版本正式发布, 同年 Dirk Merkel 发表了文章[12], 这是迄今为止有关 Docker 引用量最高的论文 (被引次数: 2 890), 掀起了学术界研究 Docker 容器技术的热潮. 然而, 由于学术界相较于工业界的滞后性, 2015 年才出现符合我们筛选标准的高质量研究论文. 此外, 虽然 2021 年发表的论文数目出现了小幅下降, 但这也并不意味着关注度的降低, 因为本文设置的检索截止时间是 2022 年 1 月, 2021 年发表的部分论文在相关数据库中还没有录入, 因此无法检索到.

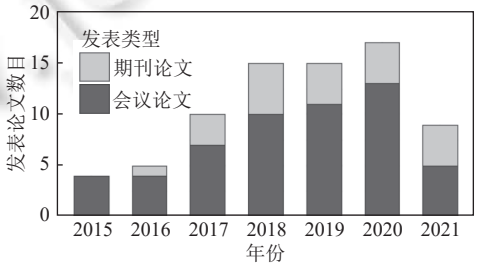


图 2 不同年份的论文发表数量分布

根据论文作者所属的研究机构, 本文进一步将论文分为国外研究、国内研究与合作研究. 当论文作者所在机构全部是国外机构时属于国外研究, 反之则为国内研究, 而当研究机构既包含国外机构又涉及国内机构时则为合作研究. 结果显示国外研究共 58 篇, 国内研究共 11 篇, 而合作研究仅有 6 篇. 这表明国内外研究者对 Docker 容器领域的关注度还存在一定差距, 国内研究者应当加大对该领域的投入和研究, 进一步开展更多的国际合作.

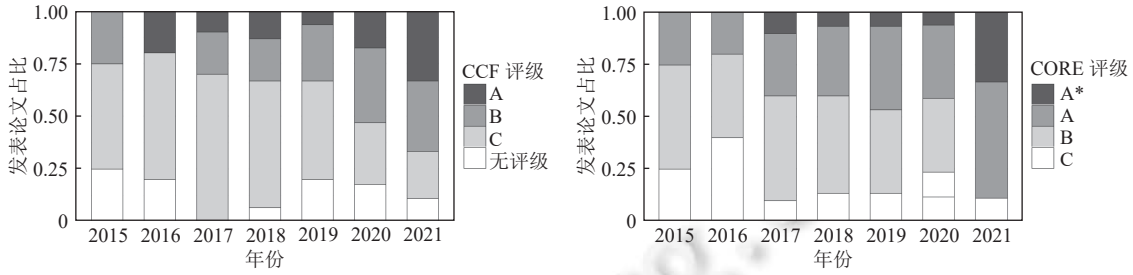
3.2 研究质量

本文对相关论文的质量评级进行了统计. 如后文图 3 所示, 不论是从国内还是国际的学术评级上看, 相关论文主要发表在高质量的学术会议/期刊上. 在 CCF 推荐期刊和会议评级中, 共有 11 篇论文属于 A 评级, 19 篇论文属于 B 评级; 在 CORE 的评级中, 共有 7 篇论文属于 A*评级, 27 篇论文属于 A 评级, 28 项论文属于 B 评级, 这表明大多数关于 Docker 研究的工作都被更高水平的期刊或会议所接收 (CCF: 40.0%; CORE: 82.7%). 同时, 随着时间推移, 评级为 C 以上论文的发表占比显著提升, 这表明 Docker 容器的研究热度在不断增加, 更高水平的期刊/会议正在加大对这个领域的关注, 该研究领域具有很高的研究价值和广阔的发展前景.

3.3 研究领域

本文采用 CCF 提出的期刊和会议领域分类, 对 75 篇论文进行了划分, 结果见表 2. 其中, 属于软件工程/系统软件/程序设计语言 (简称软件工程) 领域的期刊和会议总计达到 17 种 (占比 37.0%), 包括论文 28 篇 (占比

37.3%). 接下来, 按照各领域发文期刊和会议数量从高到低的顺序依次是: 计算机体系结构/并行与分布计算/存储系统 (简称计算机体系结构) 领域期刊和会议 15 种 (占比 32.6%), 包括论文 21 篇 (占比 28.0%); 交叉/综合/新兴领域期刊和会议 7 种 (占比 15.2%), 包括论文 19 篇 (占比 25.3%); 计算机网络领域期刊和会议 4 种 (占比 8.7%), 发表论文 4 篇 (占比 5.3%); 其他领域期刊和会议 3 种 (占比 6.5%), 发表论文 3 篇 (占比 4.0%), 该领域涉及计算机图形学与多媒体、人工智能和网络与信息安全.



(a) CCF 评级 (b) CORE 评级

图 3 不同年份发表论文的 CCF 评级以及 CORE 评级分布

表 2 不同研究领域论文发表数量分布

研究领域	期刊		会议		合计	
	期刊数	论文数	会议数	论文数	期刊/会议数	论文数
软件工程/系统软件/程序设计语言	4	8	13	20	17	28
计算机体系结构/并行与分布计算/存储系统	3	7	12	14	15	21
交叉/综合/新兴	2	2	5	17	7	19
计算机网络	2	2	2	2	4	4
其他 (计算机图形学与多媒体/人工智能/网络与信息安全)	2	2	1	1	3	3

统计结果表明, Docker 容器技术不仅成为软件工程和计算机体系结构领域的研究热点, 还受到了包括云计算、边缘计算等交叉新兴领域的广泛关注. 对研究内容进一步分析可知, 不同领域研究者的研究目标和研究视角各有不同. 例如, 软件工程领域的研究主要聚焦于对 Docker 的应用实践进行实证研究, 加深对 Docker 容器的认识和理解, 为开发者提供更多的经验证据和理论指导[13,31]; 计算机体系结构领域的研究主要关注 Docker 容器的性能比较和提升, 为加快容器的服务速度、减少能源消耗以及负载平衡等提供解决方案[32,33]; 交叉、综合以及新兴领域的研究则主要集中在云计算背景下的容器编排和资源管理[34,35]; 计算机网络领域的研究主要关注于使用微服务的容器部署以及使用容器的安全影响[36,37]; 其他领域的研究则注重与不同学科理论方法的结合, 例如基于区块链的 Docker 镜像去中心化内容信任和基于机器学习的容器化应用自动扩展[38,39].

3.4 研究方式

本文统计了论文研究方式 (包括贡献类型和评估方法) 的分布情况, 如图 4 所示. 可以看到 Docker 容器的相关研究中解决方案占主导地位 (67%), 侧重于针对具体问题提供技术解决方案. 其次是评估类型的研究 (33%), 侧重对 Docker 容器的使用情况进行评价, 或将其与虚拟机以及其他容器技术作比较, 分析 Docker 容器技术的优缺点. 然而, 其他类型的研究, 例如哲学 (侧重理论框架凝练)、观点 (侧重个人观点评述) 和经验论文 (侧重工业界经验总结) 等在该领域仍存在空白.

通过对评估方法进行统计, 本文发现控制实验是最常用的评估方法 (63%), 该方法通过对一个可测试的假设进行实验调查, 设置条件以隔离感兴趣的变量并测试它们如何影响某些可测量结果, 验证解决方案的有效性, 但其忽略了上下文因素和实际情况. 而案例研究 (29%) 作为一种详细的探索性调查技术, 主要使用定性分析方法试图理解和解释现象或检验理论, 很好地弥补了控制实验的不足. 此外, 也有小部分研究采用经验报告 (5%)、举例说

明 (2%) 和问卷调查 (1%) 的方法来评估他们的发现. 总体来看, 在 Docker 容器研究中, 研究人员会采用多样化的研究手段.

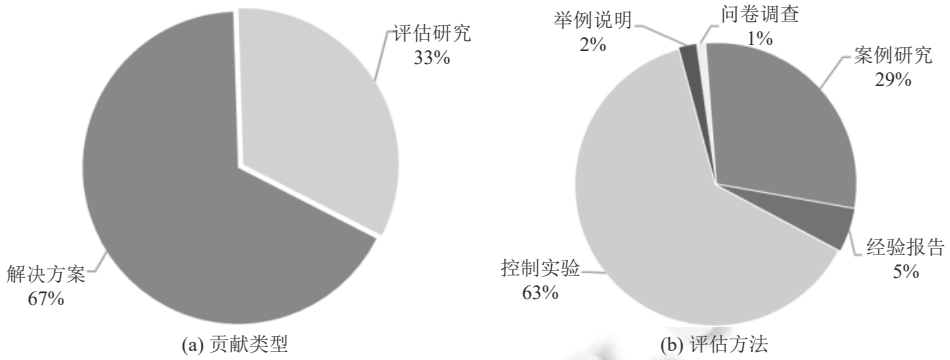


图 4 贡献类型和评估方法分布

4 Docker 容器研究主题分类

本节将阐述 Docker 容器研究的主题分类结果. 如第 2 节所述, 本文采用人工分类方法得到每篇论文的主题标签, 并通过所有作者讨论整理得到主题类别. 如表 3 所示, 本文将所有研究分为核心研究、平台研究和支持研究 3 个类别, 具体包括 8 个主题. 其中, 核心研究关注 Docker 容器的基本组件, 包括容器配置与容器镜像, 能够为平台研究和支持研究提供基础支撑; 平台研究则在核心研究基础上, 关注 Docker 容器在分布式集群环境中运行的关键技术, 涉及资源调度、集群管理和服务部署; 支持研究则关注 Docker 容器基本组件或者分布式集群运行过程中的安全性、性能等问题.

表 3 Docker 容器研究分类概述

类别	主题	论文数目	文献
核心	容器配置	12	[13,31,40–49]
	容器镜像	10	[50–59]
平台	资源调度	11	[33,60–69]
	集群管理	10	[34,35,39,70–76]
	服务部署	10	[10,36,77–84]
支持	安全分析	11	[37,38,85–93]
	性能分析	7	[6,32,94–98]
	其他	4	[99–102]

4.1 容器核心研究

容器核心研究涉及 Docker 容器运行的基本组件, 共有 22 篇 (占比 29.3%) 论文, 包括容器配置研究 12 篇和容器镜像研究 10 篇.

(1) 容器配置

容器配置定义了镜像构建时的指令执行顺序以及镜像具体架构, 通常包含多种类型的指令和参数. Docker 在 Dockerfile 中声明并运行指令, 所有指令一般都是从基础镜像开始, 然后在此基础上增加其他指令层. 其中, Dockerfile 是一个文本文件, 具体包含 17 种指令, 涉及 FROM、RUN、COPY、ADD、ENV、CMD、EXPOSE 等. 表 4 显示了 Docker 容器配置研究论文的相关信息 (按照年份排序). 目前, 实证研究是容器配置研究的重要研究方法, 大规模的实证分析能够提供对容器配置文件使用情况的整体认识. 2017 年, Cito 等人[13]首次对 GitHub 中的 7

万多个 Dockerfile 进行了探索性的实证研究, 他们发现在 560 个具有代表性的项目中, 有 34% 的 Dockerfile 没有成功构建; 最受欢迎的项目比其他 Docker 项目变化更频繁, 平均每年更改 5.81 次, 大多数 Dockerfile 的更改是为了解决构建依赖性问题; 而最常见的依赖性问题是由于配置缺少版本定义 (version pinning) 引起的.

表 4 Docker 容器配置研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[13]	2017	C	A	软件工程/系统软件/程序设计语言	评估研究
[40]	2019	B	A	软件工程/系统软件/程序设计语言	评估研究
[42]	2019	A	A*	软件工程/系统软件/程序设计语言	解决方案
[31]	2020	C	B	软件工程/系统软件/程序设计语言	评估研究
[43]	2020	B	A	软件工程/系统软件/程序设计语言	解决方案
[44]	2020	C	B	计算机体系结构/并行与分布计算/存储系统	解决方案
[45]	2020	A	A*	软件工程/系统软件/程序设计语言	解决方案
[47]	2020	B	A	软件工程/系统软件/程序设计语言	解决方案
[41]	2021	C	A	软件工程/系统软件/程序设计语言	评估研究
[46]	2021	A	A*	软件工程/系统软件/程序设计语言	解决方案
[48]	2021	A	A	软件工程/系统软件/程序设计语言	评估研究
[49]	2021	A	A*	软件工程/系统软件/程序设计语言	评估研究

在 Cito 等人研究的基础上, 后续有了更多关于容器配置的实证研究, 涵盖了 Dockerfile 的重复问题[40]、演化情况[31]和分布情况[41]等. 2019 年, Oumaziz 等人[40]对同个项目中 Dockerfile 的重复问题进行实证研究. 他们发现由于支持多个版本、基础镜像以及喜好等原因, 同个项目可能需要维护多个 Dockerfile, 而由于软件安装配置、依赖性管理和运行时配置等原因, 这其中又存在着大量重复文件. 2020 年, Wu 等人[31]对 GitHub 上托管的 4 110 个开源项目的 Dockerfile 演化情况进行实证研究, 他们调查了 Dockerfile 变化的频率、幅度和指令, 并报告了 Dockerfile 与其他文件共同变化的情况. 他们发现在初始提交后每个 Dockerfile 平均变化 6.5 次, 每次更改涉及 4.8 行代码. 特别是, RUN 指令的改变比其他指令更频繁. 同时 Dockerfile 与其他文件紧密耦合, 占所有更改的 70% 以上. 2021 年, Eng 等人[41]对超过 940 万个 Dockerfile 数据进行了实证研究, 通过分析 Docker 的变更日志, 他们发现 7.99% 的 Dockerfile 存在于一个以上的不同存储库中, 而大多数存储库包含多达 6 个 Dockerfile. 同时他们也证实了 JavaScript 是包含 Dockerfile 的项目中最流行的语言, RUN 是最流行的 Dockerfile 指令等.

近几年, 也有很多研究面向 Dockerfile 设计自动化工具和方法, 例如特定语言的环境依赖性自动推断[42]、基础镜像自动更新[43]、构建结果自动预测[44]以及 Dockerfile 的检测[45]和修复[46]. 2019 年, Horton 等人[42]提出了 DockerizeMe 工具, 将依赖性解析应用于 Python 代码片段. 他们从 Python 包索引 (PyPI) 中获取流行的 Python 资源和依赖关系的离线知识, 进一步构建 Docker 配置规范. 2020 年, Kitajima 等人[43]提出了一种自动化的基础镜像更新推荐方法, 可以根据 Git 提交的日期推断出实际使用的基础镜像版本, 当用户使用旧版本时, 能够将 Dockerfile 重构为最新版本. 同年, Wu 等人[44]提出了一种基于容器配置语义理解的容器构建结果自动预测方法 (PDBR), 该方法主要从 Dockerfile 代码中提取语义特征, 然后利用分类算法构建预测模型, 从而实现对 Docker 镜像构建结果的自动化预测. 此外, Henkel 等人[45]为 Dockerfile 创建了一个静态检查器 (binnacle), 通过提取大约 17.8 万个唯一的 Dockerfile, 他们识别了 23 个 Dockerfile 最佳配置实践规则. 在此工作的基础上, 他们在 2021 年提出了 SHIPWRIGHT 工具[46], 用于修复损坏的 Dockerfile. 该工具使用 BERT 语言模型来分析构建日志, 对损坏的 Dockerfile 进行聚类, 进一步设计出 13 条规则来对 Dockerfile 进行自动修复.

为了支持多个容器的组合应用, Docker 容器提供了 Docker Compose 服务, 开发人员可以通过自定义 docker-compose.yml 配置文件指定构成应用程序的镜像以及它们之间的交互关系. 2020 年, Piedade 等人[47]提出了一种可视化的方法来完成 docker-compose.yml 配置定义, 大大提升了易用性, 他们发现通过与纯文本标准规范比较, 使用可视化方法能够减少开发时间和出错率. 2021 年, Ibrahim 等人[48]对 4 103 个使用了 Docker Compose 的 GitHub 开

源项目进行实证研究, 结果发现应用程序并没有充分受益于 Docker Compose 的可用选项和版本: 26.8% 的项目只应用单独一个容器镜像; 应用程序只使用和维护 Docker Compose 的基本选项, 而更多的高级选项几乎被应用程序忽略; 同时很少有项目升级他们的 Docker Compose 版本. 2021 年, Ksontini 等人[49]通过人工调查定义了 Docker 项目的特定重构 (refactorings) 和技术债务 (technical debt) 类别. 他们提出了 14 个 Dockerfile 相关重构、12 个 Docker Compose 相关重构和 7 个技术债务类别. 他们发现开发人员重构这些配置文件的原因多种多样, 例如减少 Dockerfile 镜像大小、提高 docker-compose.yml 的可扩展性等.

(2) 容器镜像

Docker 镜像可以抽象为由基础镜像作为顶部的一系列数据层. 当开发者对容器进行修改时, Docker 不会直接将修改内容写入容器镜像, 而是添加一个附加层, 其中包含对镜像的修改. 表 5 显示了 Docker 容器镜像研究论文的相关信息. 2020 年, Lin 等人[50]对 3 364 529 个 Docker 镜像和 378 615 个 Git 存储库进行了大规模实证研究, 发现基础镜像更多依赖即用型语言和应用程序, 而不是尚未配置的操作系统, 且 Docker 镜像的大小呈下降趋势. 此外, 他们也发现很多用户还在使用过时的操作系统基础镜像, 而这存在质量风险. 同年, Ibrahim 等人[51]调查了 5 个软件系统和 936 个 Docker Hub 镜像, 通过实证研究比较了同一软件系统中 Docker 镜像的依赖库数量. 他们发现用户倾向于下载官方镜像 (由 Docker 本身提供的镜像) 以获得更高的资源使用效率.

表 5 Docker 容器镜像研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[56]	2017	A	A*	软件工程/系统软件/程序设计语言	解决方案
[54]	2018	A	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[57]	2018	C	B	软件工程/系统软件/程序设计语言	解决方案
[52]	2019	A	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[55]	2019	C	B	交叉/综合/新兴	解决方案
[58]	2019	B	B	交叉/综合/新兴	解决方案
[50]	2020	B	A	软件工程/系统软件/程序设计语言	评估研究
[51]	2020	A	A	软件工程/系统软件/程序设计语言	评估研究
[53]	2021	A	A*	计算机体系结构/并行与分布计算/存储系统	评估研究
[59]	2021	—	C	计算机体系结构/并行与分布计算/存储系统	评估研究

当同一个库或包在多个 Docker 容器中被使用时, 会出现磁盘空间重复问题. 2019 年, D'Urso 等人[52]提出了一种避免磁盘空间重复的方法 (Wale), 旨在收集各种 Docker 镜像之间共享的库. 用户可以在一个 Wale 文件中描述容器的特征, 然后 Wale 工具处理并创建适当的 Docker 镜像, 其中公共库和文件被移动到内存镜像中与新的镜像一起构建. 为了对依赖性错误进行及时预警, 2021 年, Gkikopoulos 等人[53]对 Docker 镜像样本进行了为期一年的系统演变分析, 提出可使用硬件、设备和体量信息来增强容器镜像元数据, 进一步他们设计了启发式硬件依赖性检测器和硬件感知的 Docker 执行器 (hdocker), 能够在产生依赖性错误时发出早期警告.

为了有效管理和存储容器镜像, Docker 提供了注册表服务 (registry). 注册表类似于一个中央镜像存储库, 允许用户发布他们的镜像, 并让其他人可以拉取 (pull) 它们. 目前, Docker 注册表有 Docker Hub、IBM 云容器注册表、Quay.ioi 和 Artifactory 等. 2018 年, Anwar 等人[54]基于 IBM 云容器注册中心超过 181.3 TB 的数据, 对注册中心的工作负载进行了全面分析, 推断出一些关于容器工作负载的关键见解, 包括请求类型分布、访问模式和响应时间等. 基于这些见解, 他们提出了有效的缓存和预取策略, 显著提高了容器性能. 2019 年, Littley 等人[55]提出、实现并评估了一种用于容器注册中心的新型框架 (BOLT). 该框架采用了一个自定义一致的哈希函数, 利用镜像的分层结构和寻址, 在不同的服务器上实现负载平衡请求.

特别地, Docker Hub 是世界上最大的容器镜像存储库, 已经托管了超过 904 万个 Docker 镜像, 总下载达到了 3 180 亿次. 目前, 已经有许多研究围绕 Docker Hub 开展分析, 例如镜像的分割[56]、标签推荐[57,58]和文件存储[59]

等. 2017 年, Rastogi 等人[56]提出了一个容器分割工具 (Cimplifier). 给定简单的用户定义约束, 该工具能够将容器划分为更简单的容器, 这些容器相互隔离并且只在必要时进行通信. 他们对真实容器的评估表明 Cimplifier 使镜像大小减少了 95%, 并且能够在 30 s 内处理大型容器. 2018 年, Yin 等人[57]提出了一种 Docker 镜像存储库的标记推荐方法 (STAR). 该方法通过解析存储库的描述和源代码, 使用 L-LDA 算法和基于相似性的排名来进行标签推荐. 在此基础上, Chen 等人[58]于 2019 年提出了一种基于半监督学习的标签推荐方法 (SemiTagRec), 通过使用累积的标记库和扩展的标记词汇表迭代地训练预测器. 2021 年, Zhao 等人[59]对 Docker Hub 注册表中存储的镜像和层进行了大规模的表征和冗余分析. 他们观察到只有 3% 的文件是唯一的, 大多数文件是可执行文件、目标文件、库、源代码和脚本, 这表明存在大量相似的镜像层, 而删除这些副本可以减少 50% 的存储空间.

小结. 目前, 容器核心研究主要关注容器配置以及容器镜像, 但尚未涉及容器架构、容器通信和容器运行时等方面. 其中, 大多数容器配置研究发表于软件工程领域 (11 篇), 且大部分论文采用实证研究方法 (6 篇), 它们侧重关注 Dockerfile 以及 docker-compose.yml 使用过程中质量问题的度量表征, 只有少数研究在实证分析的基础上进一步提出了修复方法 (2 篇). 容器镜像研究基本上产生于 2017 年之后, 侧重关注容器镜像的质量问题、空间消耗以及注册表管理, 如何优化容器镜像文件存储已经成为计算机体系结构和软件工程领域研究的热点 (各有 4 篇), 实证研究与方法研究均有涉及.

4.2 容器平台研究

容器平台研究涉及 Docker 容器在分布式集群环境中运行的相关技术, 共有 31 篇 (占比 41.3%) 论文, 包括资源调度研究 11 篇、集群管理研究 10 篇和服务部署研究 10 篇.

(1) 资源调度

在分布式集群环境中, 资源调度扮演了关键的角色, 它负责集群可用节点的工作负载分配, 并管理容器的生命周期. 表 6 显示了 Docker 容器资源调度研究论文的相关信息. 2015 年, 华为的 Li 等人[60]描述了一个基于资源网络 (RON) 概念的 REST 服务框架. 该框架将资源表示平面、控制平面和数据平面解耦, 采用自顶向下的开发思路, 通过封装底层的 Linux 资源控制模型, 提供了统一且内聚的 REST API 来管理进程、任务、作业、容器、服务器和集群中的细粒度资源. 2019 年, Huang 等人[61]开发了一个容器资源视图, 将实际资源分配信息导出到容器化应用程序中. 资源视图的中心设计是每个容器的 sys_namespace, 它可以在容器之间存在资源共享的情况下计算 CPU 和内存的有效容量. 实验结果表明, 在各种容器化应用中, 准确的资源分配视图可以获得更合适的配置并提高性能.

表 6 Docker 容器资源调度研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[60]	2015	C	B	交叉/综合/新兴	解决方案
[33]	2017	C	C	计算机体系结构/并行与分布计算/存储系统	解决方案
[68]	2017	—	C	交叉/综合/新兴	解决方案
[62]	2017	C	B	计算机网络	解决方案
[63]	2017	—	B	计算机网络	解决方案
[64]	2018	C	B	计算机体系结构/并行与分布计算/存储系统	解决方案
[65]	2018	C	B	计算机体系结构/并行与分布计算/存储系统	解决方案
[61]	2019	B	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[66]	2019	—	B	交叉/综合/新兴	解决方案
[69]	2020	C	B	交叉/综合/新兴	解决方案
[67]	2020	—	C	交叉/综合/新兴	解决方案

进一步, 研究人员提出了许多先进的资源调度算法来提升容器性能诸如响应时间[33]、负载平衡[62–64]、成本[65]、能源消耗[33,66,67]和资源利用率[66]等. 2017 年, Wu 等人[33]开发了一种确认可用的缓冲层优先调度器 (ABP), 利用 Docker Swarm 中节点的本地缓冲层来减少网络流量并加速服务启动. 通过将 ABP 算法与 Docker Swarm 中的默

认算法相比较, 实验发现 ABP 调度器将扩展服务的延迟从 10 s 以上缩短到 2 s 以下, 并在某些情况下加快了 Docker Swarm 中服务的创建. 同年, Mao 等人[62]为 Docker Swarm 中的一组异构节点设计了动态资源感知的布局方案 (DRAPS), 根据可用资源和其他服务对该节点的需求来选择部署容器的节点, 将任务分配给适当的节点并平衡异构集群中的资源使用. 实验表明, 相比于流行的容器编排工具 SwarmKit, DRAPS 具有更有效和平衡的资源使用, 但网络消耗很高. Alzahrani 等人[63]提出了一种自适应完全公平调度策略 (adCFS). 该策略能够估计不同容器工作负载特征 (如 CPU 使用情况、任务运行时、任务数等) 的权重, 并根据相应权重重新分配 CPU, 同时利用马尔可夫链模型预测 CPU 状态. 他们发现对于那些运行大型任务的容器, 容器 CPU 响应速度与默认的调度策略相比提高了 12%.

为了更好地解决容器的多目标调度问题, 2018 年 Menouer 等人[64]提出了一种新的多目标调度策略, 结合了 PROMETHEE 和 Kung 多目标决策算法, 考虑多目标条件等待的 CPU 数量和空闲内存大小, 选择一个在多目标标准之间具有良好折中性的节点来执行容器. 该策略已被应用于云计算环境, 并在 Docker SwarmKit 中实现. 同年, Zhang 等人[65]提出了一种具有成本效益和延迟感知的工作流调度算法. 该算法基于拉伸 (stretch out) 和压缩 (compact) 技术, 通过关键路径分析将任务沿资源进行拉伸, 并在虚拟机中找到低效的槽位进行任务压缩. 他们发现利用该算法, 可以在减少网络带宽消耗的同时减少总执行时间. 2019 年, Mendes 等人[66]提出了一种新型调度算法, 通过扩展 Docker Swarm 中的 Binpack 策略提高了 CPU 和内存的利用率. 在调度实际 CPU 和内存密集型工作负载 (例如视频编码、键值存储和深度学习算法) 方面, 该解决方案的成功分配率可以达到 83%, 但需要花费更多时间. 2020 年, Hamdi 等人[67]针对裸机 (bare metal) 部署模型下的容器整合问题, 提出了一种多权重容器整合策略 (MWCC), 目标是在最小数量的物理主机上根据容器权重来整合容器. 他们评估了该策略的性能, 发现该策略在保持理想服务质量 (QoS) 水平的同时, 显著降低了能源消耗.

为了支持容器组件资源在大规模云平台间动态迁移, 2017 年 Barlaskar 等人[68]提出了一个多目标动态迁移决策器 (MyMinder), 检测是否存在违反服务质量并造成性能下降的问题, 并动态地决定用户的虚拟机是否需要从当前提供商迁移到另一个提供商. 进一步, 他们提出了一个自动触发算法 (ATA), 除了 Docker Swarm 的核心自动分配功能外, 还执行虚拟机分配和取消分配功能. 实验评估表明, 他们可以在一个云提供商内将用户的应用从一个虚拟机迁移到另一个虚拟机, 而不依赖于云提供商, 并且使得迁移的开销最小. 2020 年, Xu 等人[69]提出了一个名为 Sledge 的高效动态迁移系统, 它通过在运行时迁移过程中整合镜像和管理上下文来确保组件的完整性. 他们设计了轻量级容器注册表 (LCR) 机制来实现端到端镜像迁移, 然后通过整合 CRIU 的增量内存检查点功能来迭代地迁移 Docker 容器的运行时, 最后利用 Docker 守护进程和 Docker 容器之间的耦合关系设计了动态上下文加载方案 (DCL), 精确地将 Docker 容器的管理上下文加载到正在运行的守护进程中. 实验结果显示 Sledge 减少了 57% 的总迁移时间、55% 的镜像迁移时间以及 70% 的停机时间.

(2) 集群管理

容器集群的典型结构由管理节点和工作节点组成, 其中工作节点负责运行用户提交的工作负载, 管理节点负责编排工作节点上的集群和容器部署, 不断检查并维护集群的状态. 集群架构通常由共享服务发现和编排引擎组成, 例如自动可扩展性 (auto-scalability)、负载平衡 (load balancing)、监控 (monitoring)、伸缩 (scaling) 以及文件存储 (file storage) 等. 表 7 显示了 Docker 容器集群管理研究论文的相关信息. 2016 年, Peinl 等人[70]对 Docker 集群管理的解决方案进行分类, 并将它们与案例研究中的需求进行映射, 明确了它们之间的差距. 他们用自己的集成组件和工具来弥补其中的部分差距, 形成了一个完整的管理套件, 但该管理套件无法满足所有需求.

容器支持通过动态利用水平和垂直弹性来处理工作负载波动的可能性: 水平弹性包括添加或删除与应用程序相关联的计算资源实例, 也称为资源复制; 垂直弹性是指计算资源 (CPU 时间、核数、内存、网络带宽等) 的增减特性, 也称为调整资源的大小. 这两种弹性都由工作负载需求的变化驱动, 例如请求响应时间或最终用户的数量. 2015 年, Hoenisch 等人[71]通过将决策标准表述为一个多目标优化问题来解决弹性伸缩问题, 其中虚拟机和容器可以水平调整 (实例数量的变化), 也可以垂直调整 (实例可用计算资源的变化). 他们提供了一个多目标的优化模型, 并使用优化模型为部署平台做出自动伸缩决策. 实际的应用程序评估表明该方法可以将每个请求的平均成本降低

大约 20%–28%. 为了满足对集群环境的动态灵活配置需求, 2017 年 Higgins 等人[35]在单个 HPC 集群环境下进行了虚拟容器集群 (VCC) 模型的设计、实现和测试. 他们提出了一种新的解决方案, 支持对虚拟集群的上下文感知发现和配置. 他们发现与软件定义网络 (SDN) 技术相结合, VCC 实现了透明扩展和跨越多个物理资源等动态功能. 2017 年, de Alfonso 等人[72]引入了开源的 Docker 弹性集群 (EC4Docker) 并与 Docker Swarm 集成, 在分布式部署中创建自动扩展的容器虚拟计算机集群. 他们发现与传统的虚拟机相比, 新工作节点的部署时间短, 潜在地减少了虚拟机在 CPU、内存和存储方面的开销. 在科学文献和工业实践中, 大多数提出的方法关注水平弹性, 但很少涉及垂直弹性. 2017 年, Al-Dhuraibi 等人[34]提出了一个自主驱动 Docker 容器的垂直弹性系统 (ElasticDocker). 基于著名的 IBM 自主计算 MAPE-K 原则, 该系统根据应用程序工作负载向上或向下扩展分配给每个容器 CPU 和内存. 由于垂直弹性受到主机容量的限制, 当主机上没有足够的资源时, 该系统会进行容器实时迁移. 他们的实验表明该系统有助于降低容器客户的费用, 提高容器提供商的资源利用率, 并提高应用程序最终用户的体验质量. 与水平弹性相比, ElasticDocker 的弹性性能比 Kubernetes 的要好 37.63%.

表 7 Docker 容器集群管理研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[71]	2015	B	A	软件工程/系统软件/程序设计语言	解决方案
[70]	2016	C	B	计算机体系结构/并行与分布计算/存储系统	解决方案
[34]	2017	C	B	交叉/综合/新兴	解决方案
[35]	2017	B	B	交叉/综合/新兴	解决方案
[72]	2017	B	A	软件工程/系统软件/程序设计语言	解决方案
[74]	2018	B	A	软件工程/系统软件/程序设计语言	解决方案
[73]	2019	C	B	交叉/综合/新兴	解决方案
[75]	2019	A	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[39]	2020	C	B	人工智能	解决方案
[76]	2020	A	—	计算机体系结构/并行与分布计算/存储系统	解决方案

大多数现有的弹性策略采用基于阈值的启发式方法, 也有研究提出了基于机器学习的方法来控制基于容器应用程序的弹性. 2019 年, Rossi 等人[73]提出了基于强化学习的解决方案, 控制基于容器应用程序的水平和垂直弹性, 提高应对不同工作负载的灵活性. 他们在 Elastic Docker Swarm 中集成了该方法, 通过模拟和基于原型的实验证明了该模型可以根据用户定义的部署目标成功地学习最佳的适应策略. 2020 年, Imdoukh 等人[39]提出了一种基于主动机器学习的方法来执行 Docker 容器的自动缩放, 以响应运行时的动态工作负载变化, 结果表明该架构能够逐步减少容器复制的数量, 从而使其在处理传入的工作负载急剧增加时更加高效.

在跨异构的容器编排管理与持久性存储方面, 2018 年 Brogi 等人[74]提出了一种基于云应用拓扑和编排规范 (TOSCA) 的多组件应用程序表示方法, 它允许开发人员只描述构成应用程序的组件、这些组件之间的依赖关系以及每个组件所需的软件支持. 然后, 他们提出了一个自动完成 TOSCA 应用规范的工具 (TosKeriser), 它可以自动发现基于 Docker 的运行时环境, 为应用组件提供所需的软件支持. 然而, 在不同部署平台工作负载访问的场景中, 以可伸缩的方式加载有状态的应用程序到容器编排器中并没有得到很好的支持. 为了有效地支持这些类型的工作负载, 需要使用持久性存储 (例如文件系统或块存储) 来维护数据并使其可用于容器. 2019 年, Mohamed 等人[75]提出了 Ubiquity 框架, 提供了跨异构容器编排器对持久存储的无缝访问, 并且可扩展到其他容器框架和不同类型的文件/块存储系统, 支持独立于容器编排进行管理. 2020 年, 张浩等人[76]提出了一种异构云环境下高效 Docker 镜像分发方案, 并实现了 Docker 镜像分发系统 (RainbowD). 该系统采用了一种新型的高并发机制处理镜像分发的任务, 在有限的资源下提升镜像分发效率, 解决了 P2P 网络中存在的消息延迟、消息重复和冗余等问题, 减少了带宽资源的浪费.

(3) 服务部署

容器技术在应用包装、部署和管理方面的巨大优势, 使其在生产环境中得到了广泛应用. 然而采用传统方法

的部署过程相对缓慢, 难以适应大规模的并发部署, 且中央镜像存储库上的资源争用会加剧这种困难, 部署延迟过长问题已成为容器部署的瓶颈. 表 8 是 Docker 容器服务部署研究论文的相关信息. 2016 年, Harter 等人[77]开发了一个新的容器基准 (HelloBench), 用于评估 57 个不同的容器化应用程序的启动时间. 结果显示拉包 (pull packages) 占容器启动时间的 76%, 但只有 6.4% 的数据被读取. 接着, 他们设计了一个新的 Docker 存储驱动程序 (Slacker), 可以利用专门的存储系统来实现快速容器分发. 结果显示该方法可将容器开发周期加快 20 倍, 部署周期加快 5 倍. 2018 年, Du 等人[78]提出了一个 Docker 容器部署系统 (Cider), 通过将工作节点的本地 Docker 存储改变为全节点共享的网络存储, 按需加载镜像数据. 此外, 容器的本地写时复制层可以确保实现可扩展性, 同时提高整体部署的成本效益. 实验发现在部署一个容器和一百个并发容器时, Cider 将整个部署时间平均缩短了 85% 和 62%, 能够以高并发和可扩展的方式实现快速的容器部署.

表 8 Docker 容器服务部署研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[77]	2016	A	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[83]	2016	C	B	交叉/综合/新兴	解决方案
[10]	2018	A	A*	软件工程/系统软件/程序设计语言	评估研究
[36]	2018	C	A	计算机网络	解决方案
[84]	2018	C	B	软件工程/系统软件/程序设计语言	评估研究
[78]	2018	C	B	计算机体系结构/并行与分布计算/存储系统	解决方案
[79]	2019	C	C	计算机体系结构/并行与分布计算/存储系统	解决方案
[80]	2019	—	C	交叉/综合/新兴	解决方案
[81]	2019	C	B	计算机体系结构/并行与分布计算/存储系统	解决方案
[82]	2020	A	—	软件工程/系统软件/程序设计语言	解决方案

2019 年, Gu 等人[79]对 Docker Hub 中排名前 134 的容器镜像进行了深入分析, 他们发现分层镜像的存储延迟已成为容器部署的瓶颈, 而容器冷启动只需要少量的镜像层. 基于这两个发现, 他们提出了一种非易失性内存 (NVM) 与硬盘驱动器 (HDD) 混合的 Docker 存储框架 (N-Docker). 它可以检测瓶颈层以及冷启动所需的层, 并将其存储到 NVM 中, 以便在 NVM 容量有限的情况下加快容器的启动. 实验结果表明该方法可以将容器部署速度提高 1.21 倍, 冷启动速度提高 2.96 倍. 同年, Ahmed 等人[80]通过对大型 Docker 注册中心工作负载进行分析, 证明了 Docker 镜像共享对部署时间改进的潜力. 他们还调查了分布式文件系统并讨论了它们在雾计算 (fog computing) 环境中的适用性. 基于此他们设计了 Docker 镜像共享框架, 支持在多个 Fog 节点之间共享镜像缓冲区. 实验显示该框架显著改进了缓存命中率, 使得容器的平均部署时间减少 37% 至 78%. 同年, Nguyen 等人[81]提出了只加载一次机制 (YOLO). 每当虚拟机或容器被启动时, 该机制会拦截所有的读取访问, 并直接从启动镜像中提供服务. 启动镜像是虚拟机/容器镜像的一个子集, 包含完成启动操作所需的数据块, 已经被本地存储在快速访问存储设备 (如内存、SSD 等) 上. 实验结果显示该方法使虚拟机的启动时间加快 2–13 倍, 容器的启动时间加快 2 倍. 2020 年, 陆志刚等人[82]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. 实验结果表明, 该方法可以提高 5.8 倍以上容器镜像加载速度.

然而, 大多数现有的解决方案不允许在设计时验证容器的部署性. 2016 年, Paraiso 等人[83]提出了一种基于模型驱动的 Docker 容器管理方法, 允许在设计时使用 Docker 模型进行约束验证, 而不是在执行时对容器的可部署性进行验证, 这样可以避免浪费时间和节省开发成本. 他们还计了一个图形模型驱动工具链 (Docker designer) 来设计、推理和部署容器. 此外, 也有少数研究关注容器化持续部署工作流的选择问题. 2018 年, Zhang 等人[10]对 Docker Hub/GitHub 上支持 Docker 的持续部署 (CD) 工作流程进行了大规模的实证研究. 通过定性和定量相结合的方法, 他们研究了容器化 CD 的动机、具体工作流程、需求和障碍. 他们发现了 Docker Hub 自动构建工作流程 (DHW) 与基于 CI 的工作流程 (CIW) 两个突出的工作流, 并指出开发人员在选择不同的 CD 工作流程时, 在可配

置性、简单性、要求、性能、稳定性、人员经验等方面都需要权衡.

Docker 容器支持在操作系统和库层面上进行共享, 这使得它非常适合部署微服务体系结构中的应用. 近年来, 有相关工作围绕微服务中的容器服务部署开展了研究. 2018 年, Wan 等人[36]通过研究 Docker 的特点、基于微服务应用的要求以及云数据中心的可用资源来分析应用管理问题, 提出了一个高效的通信框架 (ADMD) 和一个次优算法 (EPTA) 来确定容器放置和任务分配. 该算法以分布式、增量的方式工作, 可扩展海量物理资源和多种应用. 实验结果显示与现有策略相比, 他们提出的框架和算法提供了更多的灵活性, 并节省了更多的服务部署成本. 同年, Jha 等人[84]对同时运行异构微服务集的 Docker 容器进行了性能评估, 采用云评估实验方法 (CEEM) 进行了全面的实验, 测量了容器间 (由两个并发执行的容器引起) 和容器内 (由两个在容器内执行的微服务引起) 两种干扰. 他们发现在一个容器内执行多个微服务也是一种可行的部署选择. 而 CPU 密集型微服务与内存或磁盘密集型微服务一起运行时, 可以提供更好的性能. 内存和磁盘密集型微服务不会受到在同一容器或多个容器中运行的其他微服务的影响. 此外, 网络密集型微服务的性能会影响在同一容器中运行的其他微服务.

小结. 容器平台研究主要集中在资源调度、集群管理和服务部署方面, 绝大多数研究都是为容器编排部署提供解决方案建议 (29 篇). 当前, 大多数资源调度研究发表于交叉/综合/新兴领域 (5 篇), 这类研究主要通过改进容器调度算法或策略, 针对某个性能指标, 实现缩短响应时间、减少能源消耗、降低成本或提高资源利用率等目的, 但缺乏对不同性能指标之间的关系认识和权衡, 在提高某一性能时缺少对其他指标的评估. 而集群管理研究主要关注基于 Docker 的自动扩展和弹性伸缩, 在实际生产过程中的有效性难以得到确认, 也未集成到工具被开发者和服务提供商所采纳. 此外, 服务部署研究侧重关注于缩短部署时间、提高容器部署速度, 亦有少数工作研究部署流程选择, 但是目前仍然缺乏自动化的方法和工具以辅助开发人员实现 Docker 的高效服务部署.

4.3 容器支持研究.

容器支持研究侧重分析 Docker 容器运行过程中的安全性、性能等问题, 共有 22 篇 (占比 29.3%) 论文, 包括安全分析 11 篇、性能分析 7 篇和其他研究 4 篇.

(1) 安全分析

目前, 许多研究对 Docker 容器的安全性进行了全面深入的研究, 帮助用户了解潜在的安全风险. 表 9 是 Docker 容器安全分析研究论文的相关信息. 2018 年, Martin 等人[37]通过对 Docker 生态系统的漏洞进行分析, 研究了在典型案例中使用容器的安全影响. 他们对该领域的相关工作进行全面调查, 根据其安全类别进行分类, 确定了评估每个组件安全的关键因素. 此外, 他们识别了 Docker 环境的不同组件中由于设计或一些原始用例引起的几种漏洞. 他们详细说明了这些漏洞可能被利用的真实场景, 并提出了可能的修复方法. 同时, 他们发现 Docker 的许多漏洞是由于将容器作为虚拟机使用造成的.

表 9 Docker 容器安全分析研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[89]	2015	—	C	交叉/综合/新兴	解决方案
[37]	2018	C	C	软件工程/系统软件/程序设计语言	评估研究
[38]	2018	C	B	计算机网络	解决方案
[90]	2018	B	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[85]	2019	B	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[91]	2019	—	B	软件工程/系统软件/程序设计语言	解决方案
[92]	2019	—	C	交叉/综合/新兴	解决方案
[88]	2020	B	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[86]	2021	A	A	计算机体系结构/并行与分布计算/存储系统	解决方案
[87]	2021	B	A	软件工程/系统软件/程序设计语言	解决方案
[93]	2021	A	A	交叉/综合/新兴	解决方案

2019 年, Zerouali 等人[85]对基于 Debian Linux 的 7380 个官方和社区 Docker 镜像进行了大规模实证研究, 识别了其中安装的软件包, 并从版本更新、安全漏洞和错误方面衡量其技术滞后性. 他们发现, 没有一个版本是没有漏洞的, 即使部署了最新的软件包也无法避免漏洞. 他们为容器开发人员提供了一些经验教训, 让他们可以遵循一些策略来最小化漏洞数量. 之后, 他们又对 140498 个 Debian 镜像的技术滞后问题进行了为期 3 年的大规模实证研究[86], 并根据 5 个维度 (软件包滞后、时间滞后、版本滞后、错误滞后和漏洞滞后) 将技术滞后框架实例化. 他们的研究表明技术滞后的不同维度是互补的, 并且在所有滞后维度中, 官方 Debian 镜像的滞后性始终低于社区镜像, 而产生的滞后量取决于 Debian 的类型和考虑的滞后维度. 接着, 他们对 3000 个流行的社区镜像中安装的 JavaScript、Python 和 Ruby 包的安装实践、过时程度和漏洞进行实证研究[87]. 结果显示在许多情况下, 这些安装的软件包是由于错过重要的版本更新导致镜像存在潜在漏洞. 他们的发现表明, Docker Hub 社区镜像的维护者应该投入更多的精力来更新这些镜像中包含的过时软件包, 以减少漏洞数量. 除此之外, 他们发现 Python 社区镜像通常比 NodeJS 和 Ruby 社区镜像的过时程度要低、受漏洞影响小.

为了揭示 Docker 镜像的安全风险来源, 2020 年 Liu 等人[88]通过大规模的实证分析发现了 Docker 镜像的 3 个主要安全风险来源, 包括运行命令中的敏感参数、Docker 镜像中执行的恶意程序以及未修补的软件漏洞. 他们基于 Docker Hub 上 975858 个存储库中的 2227244 个镜像, 研究发现带有敏感参数的运行命令会对用户和主机造成灾难性危害, 如主机文件和显示的泄露、主机拒绝服务攻击等. 他们发现了 42 个恶意镜像, 可以造成远程代码执行和恶意加密等攻击. 同时, Docker 镜像中的软件漏洞补丁存在明显延迟, 而这个现象往往被忽略.

此外, 一些研究提出了容器的安全策略、方法和技术框架. 2015 年, Zhang 等人[89]对部署在多用户环境中的应用程序容器的企业安全策略管理进行了分析. 他们介绍了一个名为 Harbormaster 的系统, 通过对 Docker 容器管理操作实施策略检查, 并允许管理员实现最小特权原则来解决问题. 该系统基于代理的架构实现了与 Docker 生态系统的透明整合, 并能够在多用户环境中执行有用的策略. 2018 年, Xu 等人[38]提出了一种基于区块链的去中心化 Docker 信任方法 (DDT), 通过区块链来分散信任. 该方案大大降低了拒绝服务攻击 (DoS) 的风险, 同时为 Docker 镜像提供了签名验证服务. 实验结果证明了该方法的可扩展性和效率, 并在跨多个数据中心的亚马逊网站服务 (AWS) 上进行了性能评估.

2018 年, Tak 等人[90]设计和实现了一个用于容器安全分析的数据处理框架. 它将数据收集过程从分析中分离出来, 以便用户更多地关注构建新的分析逻辑, 而不是关注监视代理的工具. 他们利用该框架对 Docker Hub 的前 10000 个公共容器镜像进行了广泛的实证研究, 发现超过 92% 的镜像包含违规或易受攻击的软件包. 2019 年, Fernandez 等人[91]提出了一种策略来加强云中的数据安全. 他们创建了安全容器编排器 (SCO), 利用最新的基于硬件的可信执行环境技术 (Intel SGX) 来进行数据保护. 该方法实现了 SGX 友好的容器自动扩展方案, 利用负载平衡路由和封装 SGX 兼容等特性使可信应用程序的部署与标准云实践保持一致. 他们还将 SCO 与 Kubernetes 的现状进行比较, 展示了使用这种安全增强解决方案的内在成本, 对于管理在云中使用敏感数据的应用程序, SCO 可以成为一个有效的选择.

为了确保容器化环境的正常使用, 2019 年, Gantikow 等人[92]研究了基于规则的安全监控方法. 该方法可以用来检测容器环境中工作负载的各种潜在恶意行为, 还能够监测实际容器运行时, 防止误用和配置错误. 他们发现提出的方法在许多场景中都是有效的. 2021 年, Simonsson 等人[93]提出了一种新的故障注入系统 (ChaosOrca), 它可以主动向容器化应用程序注入系统调用故障, 以评估其在面对此类扰动时的弹性. 他们详尽地分析了所有类型的系统调用, 并利用不同级别的监视技术来推断扰动下的行为. 通过对 3 个不同的真实世界应用 (文件传输客户机、反向代理服务器和面向微服务的 Web 应用程序) 进行评估, 他们发现系统调用级别的扰动有希望检测出这些应用中嵌入的自我保护和弹性机制的弱点.

(2) 性能分析

此类研究主要关注 Docker 容器的性能分析. 表 10 显示了 Docker 容器性能分析研究论文的相关信息. 现有的基准测试方法非常耗时, 因为它们通常对整个虚拟机 (VM) 进行基准测试, 以生成准确的性能数据, 这使得它们不太适合实时分析. 2016 年, Ruan 等人[94]通过一系列实验来衡量应用容器 (Docker) 和系统容器 (LXC) 之间的性能

差异, 评估了裸机和容器之间额外虚拟机层的开销, 并检查了亚马逊容器服务 (ECS) 和谷歌容器引擎 (GKE) 的服务质量. 实验结果表明系统容器比应用程序容器更适合维持 I/O 约束的工作负载, 因为应用程序容器由于文件的分层而遭受较高的 I/O 延迟. 在虚拟机中运行容器将导致严重的磁盘 I/O 性能下降 (42.7%), 网络延迟高达 233%. 同年, Varghese 等人[95]提出基于 Docker 容器的轻量级基准测试工具 (DocLite), 它将基准数据分为 4 组, 即内存和处理、本地通信、计算和存储. 用户提供一组 4 个权重 (范围从 0 到 5) 作为输入, 表明每组数据对需要部署到云上的应用程序的重要性. 这些权重被映射到 4 个基准组上, 并被用来生成一个分数, 以便根据性能对虚拟机进行排名. 他们发现使用 DocLite 进行基准测试比对整个 VM 进行基准测试的速度快 19–91 倍, 平均准确率超过 90%, 这证明了该轻量级方法的实时性和可靠性.

表 10 Docker 容器性能分析研究论文相关信息

文献	年份	CCF评级	CORE评级	领域	方法
[94]	2016	—	C	计算机体系结构/并行与分布计算/存储系统	评估研究
[95]	2016	C	C	计算机体系结构/并行与分布计算/存储系统	解决方案
[32]	2017	A	A	计算机体系结构/并行与分布计算/存储系统	评估研究
[97]	2018	C	B	交叉/综合/新兴	评估研究
[96]	2018	B	A	软件工程/系统软件/程序设计语言	评估研究
[6]	2019	A	A	计算机体系结构/并行与分布计算/存储系统	评估研究
[98]	2020	C	B	交叉/综合/新兴	评估研究

2017 年, Kozhirbayev 等人[32]提出了云虚拟化技术综合性能评估方法, 特别关注 Docker 和 Flockport (LXC). 他们评估了两者的优缺点和总体性能 (CPU、内存、网络带宽和延迟以及存储开销), 发现两者存在许多共同模式. 例如, Docker 和 Flockport 在内存和 CPU 的使用上几乎没有任何开销, 而 I/O 和操作系统交互产生了一些开销. 此外, Docker 还包括一些其他功能, 比如网络地址转换. 然而, 这些功能直接影响了吞吐量质量的输入. 因此, 在某些情况下, 没有使用额外特性的 Docker 容器不会比 Flockport 更快. 2018 年, Santos 等人[96]比较了 3 个应用工作负载 WordPress、Redis 和 PostgreSQL 在 Docker 和裸机 Linux 上的能耗. 结果发现, 在所有情况下, 在 Docker 中运行测试的能耗都更高, 这主要是由于 I/O 系统调用的性能问题. 因此, 担心 I/O 开销的开发者可以考虑裸机部署而不是 Docker 容器部署.

同时, 由于云平台倾向于在虚拟机中运行容器, 因此公共容器服务的架构也备受争议. 从性能的角度来看, 在裸机和容器之间额外的虚拟机层可能会带来不必要的性能开销. 2019 年, Mavridis 等人[6]在 KVM 和 XEN 虚拟机上运行了 Docker 容器, 还在 Windows 服务器上运行 Linux 容器并评估其性能. 他们通过执行各种基准测试和部署现实世界的应用程序作为用例, 来量化虚拟机的额外虚拟化层所带来的性能开销. 他们发现在虚拟机上运行容器的 CPU 和内存性能开销不大, 尤其是在 XEN 虚拟机上, 而磁盘和网络性能受虚拟机的额外虚拟化层影响较大. 与其他管理程序相比, KVM 管理程序取得了最高的磁盘和网络性能. 在操作系统方面, Alpine、CoreOS 和 RancherOS 取得了最高的性能. 在虚拟机上运行容器的整体性能受到应用程序类型、虚拟机数量、容器数量、客户的操作系统、存储机制和不同类型工作负载的影响. 当虚拟机和并行运行的容器数量越来越多时, 虚拟机引入了明显的开销. 2018 年, Garg 等人[97]介绍了将基于虚拟机的工作负载迁移到容器平台时遇到的问题和挑战. 他们将当前托管在虚拟机上的应用程序工作流重新转换到容器平台上, 并基于资源共享、并发性、隔离和可靠性参数进行研究. 实验发现, 对于需要运行时间隔离的动态工作负载来说, 容器在内存上的表现优于虚拟机. 2020 年, Yoshimura 等人[98]提出了一个容器性能评估框架 (ImageJockey), 支持周期性的镜像构建、简单的容器编排、指标收集和结果可视化. 他们通过分析 16 个容器镜像证明了该框架的有用性, 并指出所选容器镜像的性能变化是不可忽略的.

(3) 其他

除上述研究外, 本文还发现 4 篇论文属于容器支持研究. 2015 年, Affetti 等人[99]提出了一套用于创建高性能、

沙箱化、可配置的云基础设施实验环境的工具 (aDock). 它基于 OpenStack 和 Docker 开发, 提供了一套模拟工具为虚拟机放置和服务器整合算法创建实验环境. 2018 年, Zhang 等人[100]定义了一个“容器-虚拟机-物理机 (container-vm-pm)”架构, 并基于所涉及的 3 个实体提出了一种新的容器放置策略. 此外, 他们还建立了一个适合度函数, 用于虚拟机和物理机的选择. 实验表明, 该方法在物理资源利用率方面优于现有策略. 为了获取开发者对 Docker 技术的看法, 2020 年 Haque 等人[101]对 Stack Overflow (SoF) 社区的帖子进行了大规模的实证研究. 他们使用主题建模的方法从 Docker 相关帖子中挖掘出 30 个主题, 包括基本概念、Web 服务器配置、框架开发、应用部署、持续集成、网络和存储库等. 同年, Gholami 等人[102]提出了 Docker 框架的一个开源扩展 (DockerMV), 以支持容器软件系统的多版本控制. 通过在微服务参考测试应用程序 (TeaStore) 和容器化新闻门户应用程序 (Znn) 上进行实验, 他们证明了多版本控制在满足容器化软件系统性能需求方面的有效性.

小结. 当前容器支持研究主要包括安全分析、性能分析和其他研究, 容器监控、数据管理和日志管理等方面尚未涉及. 安全分析研究是容器支持研究的一个重要方向 (11 篇), 目前该领域绝大多数为国外学者进行的研究 (10 篇), 仅有 1 篇为国内外合作研究, 因此国内研究人员对容器安全性的研究有所欠缺. 此外, 许多研究表明 Docker 镜像包含了许多高风险漏洞, 少数工作研究了 Docker 镜像的漏洞评估工具, 然而这些工具在生产环境中的可用性, 以及它们的使用是否会阻碍部署过程还未得到确认. 性能分析方向的研究主要发表于计算机体系结构领域 (4 篇), 相关研究不仅将 Docker 与其他容器技术做比较, 还进一步将容器技术与虚拟机技术做比较, 研究表明 Docker 容器在许多方面都具有优势, 作为应用程序管理框架具有更高的灵活性, 但也存在一定的缺陷, 例如引入 I/O 和操作系统交互产生的开销.

5 研究讨论

本节旨在对 Docker 容器技术的发展趋势进行分析, 进一步对未来研究方向进行讨论.

5.1 发展趋势分析

根据不同论文的研究主题和出版年份, 本文首先描绘了不同年份的研究主题分布情况, 如图 5 所示 (气泡中数字代表论文数目). 本文发现早期研究 (2015–2017 年) 每年涉及的研究主题有 3–5 个, 而近期研究 (2018–2020 年) 每年涉及的研究主题已经增加到 7–8 个, 说明 Docker 容器研究主题随着时间发展呈现多样化的趋势. 此外, 早期研究主要侧重在资源调度和集群管理主题 (论文数目≥3), 近期研究的关注点则向服务部署、安全分析、容器镜像、容器配置主题迁移. 本文认为随着容器编排/调度工具的日益流行, 大量的资源调度和集群管理工作可以利用 Kubernetes、Amazon ECS、Google GKE 等平台级工具完成, 开发者可以将更多的注意力集中在上层应用的构建、交付和管理上, 这也使得容器配置、容器镜像、服务部署主题的研究变得越来越重要. 同时, Docker 容器的安全性问题日益受到关注, 如何准确地识别安全漏洞并快速修复成为亟待探索的问题.

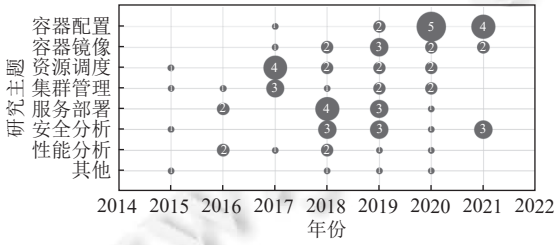


图 5 不同年份的研究主题分布气泡图

从 Docker 容器研究主题的发展趋势可以看出, Docker 容器技术在不断发展演变. 近年来, 容器运行时、镜像格式、编排工具、宿主机操作系统等都有多种选项, 开发者可以根据需求选择不同的工具或平台, 这也使得不同容器技术厂商之间的商业竞争越来越激烈. 例如, 2014 年 6 月 Google 推出 Kubernetes, 迅速占领了容器编排市场, 它作为 Docker 容器技术的推动者, 驱动了 Docker 容器运行时在工业界的广泛使用. 然而, Docker 容器公司的封闭

与运营不佳, 使得 Docker 社区与 Kubernetes 社区逐渐松绑, 形成了容器技术之间的竞争态势. 本文认为容器技术之争背后蕴含着标准之争, 容器厂商希望建立自己的标准以获得容器技术领域的话语权和影响力, 进而获得商业利益. 抛开商业因素, 本文认为 Docker 容器技术作为容器技术的开创者, 仍然是众多容器应用的基础, 在容器技术领域扮演着非常重要的作用. 虽然目前已有许多工作对 Docker 容器进行了探索, 但是相比于工业界广泛开展的应用实践, 学术界关于 Docker 容器的研究尚处于初期阶段, 还需要更多的研究与探索.

5.2 未来研究展望

本文根据对相关研究的梳理分析, 总结出了 7 个 Docker 容器领域未来值得探索的研究方向.

(1) Docker 容器配置自动化管理. 现有 Docker 容器配置脚本的管理主要依赖开发人员人工操作, 对开发人员的经验和能力提出了极高的要求[31]. 此外, 目前研究侧重于对容器配置脚本质量[13]、演化情况[31]等的度量表征进行分析, 缺乏相关的应对机制以及自动化管理方法. 为了有效提升 Docker 容器配置质量, 需要将理论分析与技术实践相结合, 建立多维度、自动化的配置管理技术体系. 以下研究内容值得进一步探索.

● 配置模式挖掘: 当前大量的 Docker 容器应用实践产生了海量的配置数据, 未来研究可以在基础度量统计的基础上, 深入挖掘不同 Docker 配置指令之间的组合模式, 从中提炼出开发者进行 Docker 容器配置的最佳实践, 从而为广大软件从业者提供有价值的 Docker 容器指令配置指导.

● 配置缺陷修复: 目前工业界仍缺乏针对 Docker 配置缺陷的自动化修复方法, 未来研究可以结合 Docker 官方文档、技术博客、相关文献和企业实践等, 扩展和细化现有的配置缺陷质量问题类型, 总结凝练出 Docker 配置反模式 (anti-pattern), 并据此研究细粒度的配置缺陷检测方法和修复方法.

● 配置自动生成: 当前 Docker 容器主要依靠人工配置, 未来研究可以在 Docker 配置指令代码表征理解的基础上, 进一步探索 Docker 配置的自动化生成方法, 从源代码、包/库中挖掘依赖信息, 结合配置模式自动化地生成 Docker 容器配置.

(2) Docker 容器镜像重构与推荐. Docker 镜像的结构比传统应用更加复杂, 一个镜像可能包含大量的程序、环境变量和配置文件, 传统的分析方法很难按比例扫描所有的镜像. 目前 Docker 容器镜像相关研究主要关注其空间消耗[52,53,59]、标签推荐[57,58]等问题, 尚缺乏针对容器镜像重构与推荐的研究, 本文认为以下研究内容值得进一步探索.

● 冗余文件识别: Docker 容器镜像中会包含各类文件, 包括基础镜像文件、依赖包/库文件、相关数据等, 未来研究可以对不同类型的 Docker 容器镜像文件内容进行分析, 并结合它们的功能特点识别出那些冗余的文件类型, 进一步可以利用“.dockerignore”文件设置避免向镜像中传输非必要的文件内容.

● 镜像图层重构: Docker 容器镜像由多个镜像图层构成, 图层的具体内容以及图层间的组合顺序对于最终镜像的架构和性能影响巨大, 未来研究可以围绕镜像图层指令优化和图层组合优化展开探索, 结合最佳配置实践对已有镜像图层进行重构.

● 基础镜像推荐: 基础镜像在 Docker 容器镜像构造中扮演着重要角色, 其空间大小和性能优劣会直接影响最终镜像的大小和性能. 未来研究可以研究自动化的基础镜像推荐方法, 根据具体 Docker 配置指令, 综合其语义信息和项目功能, 为开发者推荐合适的基础镜像.

(3) Docker 容器资源调度优化. 目前 Docker 容器资源调度相关研究提出了一系列方法和策略[62−67], 但由于 Docker 容器调度问题是 NP 难问题, 对于大型问题, 没有多项式复杂度算法来求解最优调度问题. 因此, 大多数相关研究都是应用一些启发式方法来寻找问题的近似解决方案. 启发式算法通常具有较低的复杂度, 并能在合理的时间内生成令人满意的调度. 如前所述, 已经提出的几种启发式方法大多使用 binpacking 技术, 或者是 Docker Swarm 和 Kubernetes 中现有技术的组合, 将容器映射到计算节点. 启发式通常是可扩展的, 但解决方案的最优性不能保证. 当前仍留有许多值得进一步探索的研究内容.

● 资源调度流程追踪: 现有 Docker 容器资源调度研究主要侧重实现具体的调度方法, 缺乏整个调度流程的追踪, 未来研究可以进一步分析 Docker 容器资源调度过程中的 CPU 使用情况、任务运行时、任务数等的变化情况,

从而及时发现异常.

● 高效多目标资源调度: 当前多目标资源调度研究仍然存在解决方案生成时间长[66]、能源消耗[62]等问题, 未来研究可以对高效的多目标资源调度算法展开研究, 在维持调度性能的同时, 降低能源消耗, 从而进一步形成相关工具或服务.

● 云际间资源迁移: Docker 容器组件资源在多个云平台间的动态迁移已经成为迫切需要解决的问题, 未来研究可以充分结合云计算和容器技术特点, 研发高效能的云际间资源迁移方法, 在保证资源迁移完整性的同时, 减少迁移成本.

(4) Docker 容器集群高效编排管理: 当前相关研究[70−76]为 Docker 容器集群管理提供了初步的解决思路, 但以下研究内容仍需要进一步探索.

● 集群架构适配: Docker 容器集群的架构组织对于能否发挥集群的整体效能至关重要, 未来研究可以分析不同节点的功能特点和运行模式, 研究不同类型集群、不同业务需求场景下的容器集群架构组织模式, 分析不同组织模式的内在运行机制, 进而设计实现自动化的集群架构方案推荐方法.

● 组件依赖关系解析: Docker 容器集群中的组件节点间存在复杂多样的依赖关系, 未来研究可以从相关配置和源代码中抽取组件之间的依赖关系, 并考虑时间因素, 观察节点依赖关系的演变规律. 此外, 可以设计实现相关节点重要性评估方法, 识别集群中的关键节点.

● 异构云环境集群管理: 跨异构云环境的容器集群管理面临现实需求, 未来研究可以探索异构云环境的容器编排方法, 将异构云环境的特征纳入考量, 对现有的容器编排方法进行优化和完善, 支持开发者便捷、高效地进行异构云环境中的容器集群管理.

(5) Docker 容器服务部署评估与调控: Docker 容器应用最终需要部署才能体现其应用价值, 目前研究在缩短部署时间、提高容器部署速度、部署流程选择等方面已经开展了相关探索[10,79−82], 但在效能评估、异常分析和调控等方面仍然存在诸多不足, 本文认为未来研究可以关注以下几个方面.

● 部署效能评估: 针对 Docker 容器服务部署效能的量化评估需求, 未来研究可以探索多维度的部署效能度量体系, 综合部署延迟、启动时间、稳定性、安全性等指标, 为开发者进行高效的容器服务部署、优化现有容器服务部署流程提供参考依据.

● 异常归因分析与调控: 未来研究可以对丰富的容器镜像部署现场日志数据进行挖掘, 分析失败、长延迟等异常部署记录的诱发原因, 从中总结提炼关键影响要素, 进一步可以探索自动化的异常部署结果预警方法和关键要素调控方法. 此外, 未来研究可以探索更加高效的资源监控和日志分析工具.

● 微服务部署管理: 针对当前容器应用微服务化部署的现实场景, 未来研究可以探索微服务场景下的容器部署问题, 结合已有实现案例分析高效的容器微服务部署方法, 尤其对于网络密集型微服务, 未来研究可以对其性能优化展开深入探索.

(6) Docker 容器安全漏洞体系与修复: 目前 Docker 镜像漏洞修补过程存在明显延迟, 原因是 Docker 镜像程序需要与主流程序解耦, 开发人员更新 Docker 镜像程序的积极性会较低. 安全漏洞日益成为影响 Docker 容器使用的关键问题, 当前研究提出了一系列的安全策略[37,89,90,92], 但尚未形成体系, 并且缺乏相关的漏洞预警和问题修复方法, 以下内容值得研究者们关注.

● 安全策略体系: 在现有安全策略的基础上, 未来研究可以考虑更加广泛的应用案例, 梳理形成多维度的安全策略体系, 为开发者提供更加全面准确的安全策略指南.

● 安全漏洞预警与修复: 未来研究应该考虑到不同的应用程序需求和平台的可用性、实用性、自动化、简单性和易用性. 此外, 还需要进一步研究自动检测和修复容器漏洞的可行性, 例如通过分析各类容器资源数据, 从中检测各种潜在的安全漏洞信息, 进一步可以研究相应的安全问题修复方法, 实现对于安全漏洞的自动化预警和修复.

(7) Docker 容器自身性能优化与扩展. 随着容器技术的广泛应用, 容器可以在单个设备上运行, 也可以部署在这些设备的集群上, 使其适用于边缘计算和物联网计算[17,18], 容器平台技术能够让容器作为集群在分布式环境中

运行. 随着 Docker 容器应用领域和场景的扩展, 容器虚拟化性能以及服务扩展等问题急需方法和工具的支持.

● 虚拟机中容器运行优化: 目前虚拟机中的容器运行面临严重的磁盘 I/O 性能下降问题[32,94,96], 未来研究可以综合应用程序类型、虚拟机数量、容器数量、客户的操作系统、存储机制和不同类型工作负载等因素, 探索容器 I/O 开销优化方法.

● 容器服务扩展工具: 目前 Docker 容器在工业界已经广泛应用, 未来研究可以探索面向不同场景的容器服务扩展工具, 从而更好地支持 Docker 容器的运行和管理.

6 总 结

随着软件技术的发展, 软件构造、运行和演化过程产生了诸多变化, 面临着开发测试环境需要高效切换或配置、应用隔离、减少资源消耗、提高测试和部署效率等新需求, 给开发人员开发和维护软件带来了巨大的负担. Docker 作为目前最流行的容器虚拟化技术, 它能够将应用程序的代码和依赖打包到一个轻量级、独立且可移植的执行环境中, 近些年已成为一个热门的研究领域, 众多关于 Docker 容器不同方面的研究被先后提出. 为了帮助研究人员全面准确地理解当前 Docker 容器研究的现状和趋势, 本文首次对现有 Docker 容器研究进行了系统性的综述研究. 本文对 2014 年至今的 75 篇 Docker 容器研究论文进行了系统整理和详细分析, 提出了面向 Docker 容器研究的分类框架, 并基于该分类框架对不同研究主题的论文进行归纳梳理. 最后, 本文讨论了 Docker 容器技术的发展趋势并总结出了 7 个值得探索的研究方向, 对未来的研究具有指导意义.

References:

[1] Anderson C. Docker [software engineering]. IEEE Software, 2015, 32(3): 102–105. [doi: 10.1109/MS.2015.62]

[2] Boettiger C. An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review, 2015, 49(1): 71–79. [doi: 10.1145/2723872.2723882]

[3] Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 2014, 1(3): 81–84. [doi: 10.1109/MCC.2014.51]

[4] Chae M, Lee H, Lee K. A performance comparison of linux containers and virtual machines using Docker and KVM. Cluster Computing, 2019, 22(S1): 1765–1775. [doi: 10.1007/s10586-017-1511-2]

[5] Zhang Y, Yin G, Wang T, Yu Y, Wang HM. An insight into the impact of Dockerfile evolutionary trajectories on quality and latency. In: Proc. of the 42nd Annual Computer Software and Applications Conf. (COMPSAC). Tokyo: IEEE, 2018. 138–143. [doi: 10.1109/COMPSAC.2018.00026]

[6] Mavridis I, Karatza H. Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing. Future Generation Computer Systems, 2019, 94: 674–696. [doi: 10.1016/j.future.2018.12.035]

[7] Shirinbab S, Lundberg L, Casalicchio E. Performance evaluation of containers and virtual machines when running Cassandra workload concurrently. Concurrency and Computation:Practice and Experience, 2020, 32(17): e5693. [doi: 10.1002/cpe.5693]

[8] Kratzke N, Quint PC. Understanding cloud-native applications after 10 years of cloud computing—A systematic mapping study. Journal of Systems and Software, 2017, 126: 1–16. [doi: 10.1016/j.jss.2017.01.001]

[9] Fokaefs M, Barna C, Veleda R, Litoiu M, Wigglesworth J, Mateescu R. Enabling DevOps for containerized data-intensive applications: An exploratory study. In: Proc. of the 26th Annual Int'l Conf. on Computer Science and Software Engineering. Toronto: ACM, 2016. 138–148.

[10] Zhang Y, Vasilescu B, Wang HM, Filkov V. One size does not fit all: An empirical study of containerized continuous deployment workflows. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE). Lake Buena Vista: ACM, 2018. 295–306. [doi: 10.1145/3236024.3236033]

[11] Singh C, Gaba NS, Kaur M, Kaur B. Comparison of different CI/CD tools integrated with cloud platform. In: Proc. of the 9th Int'l Conf. on Cloud Computing, Data Science & Engineering. Noida: IEEE, 2019. 7–12. [doi: 10.1109/CONFLUENCE.2019.8776985]

[12] Merkel D. Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014, 2014(239): 2.

[13] Cito J, Schermann G, Wittern JE, Leitner P, Zumberi S, Gall HC. An empirical analysis of the Docker container ecosystem on GitHub. In: Proc. of the 14th Int'l Conf. on Mining Software Repositories (MSR). Buenos Aires: IEEE, 2017. 323–333. [doi: 10.1109/MSR.2017.67]

[14] Dua R, Raja AR, Kakadia D. Virtualization vs. containerization to support PaaS. In: Proc. of the Int'l Conf. on Cloud Engineering.

Boston: IEEE, 2014. 610–614. [doi: 10.1109/IC2E.2014.41]

[15] Flexera. 2021 state of the cloud report. 2021. https://www.flexera.com/about-us/press-center/flexera-releases-2021-state-of-the-cloud-report

[16] Casalicchio E, Iannucci S. The state-of-the-art in container technologies: Application, orchestration and security. Concurrency and Computation: Practice and Experience, 2020, 32(17): e5668. [doi: 10.1002/cpe.5668]

[17] Morabito R, Farris I, Iera A, Taleb T. Evaluating performance of containerized IoT services for clustered devices at the network edge. IEEE Internet of Things Journal, 2017, 4(4): 1019–1030. [doi: 10.1109/JIOT.2017.2714638]

[18] Do Espírito Santo W, de Souza Matos Júnior R, de Ribamar Lima Ribeiro A, Silva DS, Santos R. Systematic mapping on orchestration of container-based applications in fog computing. In: Proc. of the 15th Int'l Conf. on Network and Service Management (CNSM). Halifax: IEEE, 2019. 1–7. [doi: 10.23919/CNSM46954.2019.9012677]

[19] Simonis I. Container-based architecture to optimize the integration of microservices into cloud-based data-intensive application scenarios. In: Proc. of the 12th European Conf. on Software Architecture: Companion Proc. Madrid: ACM, 2018. 1–3. [doi: 10.1145/3241403.3241439]

[20] Budgen D, Brereton P. Performing systematic literature reviews in software engineering. In: Proc. of the 28th Int'l Conf. on Software Engineering (ICSE). Shanghai: ACM, 2006. 1051–1052. [doi: 10.1145/1134285.1134500]

[21] Kitchenham B. Procedures for performing systematic reviews. Technical Report, Keele: Keele University, 2004.

[22] Chen LP, Babar MA, Zhang H. Towards an evidence-based understanding of electronic data sources. In: Proc. of the 14th Int'l Conf. on Evaluation and Assessment in Software Engineering. Keele: ACM, 2010. 135–138.

[23] Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of the 18th Int'l Conf. on Evaluation and Assessment in Software Engineering. London: ACM, 2014. 38. [doi: 10.1145/2601248.2601268]

[24] Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software, 2007, 80(4): 571–583. [doi: 10.1016/j.jss.2006.07.009]

[25] Pahl C, Brogi A, Soldani J, Jamshidi P. Cloud container technologies: A state-of-the-art review. IEEE Trans. on Cloud Computing, 2019, 7(3): 677–692. [doi: 10.1109/TCC.2017.2702586]

[26] Docker. Docker overview. 2023. https://docs.docker.com/get-started/overview/

[27] Tony. Docker ecosystem one: Container core technology. 2022. https://blog.devgenius.io/docker-ecosystem-one-container-core-technology-28bf5bbf795e

[28] Tony. Docker ecosystem two: Container platform technology. 2022. https://blog.devgenius.io/docker-ecosystem-two-container-platform-technology-4f0714101c99

[29] Tony. Docker ecosystem three: Container support technology. 2022. https://blog.devgenius.io/docker-ecosystem-three-container-support-technology-6c2ff98dbdd6

[30] Viera AJ, Garrett JM. Understanding interobserver agreement: The Kappa statistic. Family Medicine, 2005, 37(5): 360–363.

[31] Wu YW, Zhang Y, Wang T, Wang HM. Dockerfile changes in practice: A large-scale empirical study of 4110 projects on GitHub. In: Proc. of the 27th Asia-Pacific Software Engineering Conf. (APSEC). Singapore: IEEE, 2020. 247–256. [doi: 10.1109/APSEC51365.2020.00033]

[32] Kozhirbayev Z, Sinnott RO. A performance comparison of container-based technologies for the cloud. Future Generation Computer Systems, 2017, 68: 175–182. [doi: 10.1016/j.future.2016.08.025]

[33] Wu Y, Chen HP. ABP scheduler: Speeding up service spread in Docker swarm. In: Proc. of the 2017 IEEE Int'l Symp. on Parallel and Distributed Processing with Applications and the 2017 IEEE Int'l Conf. on Ubiquitous Computing and Communications (ISPA/IUCC). Guangzhou: IEEE, 2017. 691–698. [doi: 10.1109/ISPA/IUCC.2017.00108]

[34] Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P. Autonomic vertical elasticity of Docker containers with ElasticDocker. In: Proc. of the 10th IEEE Int'l Conf. on Cloud Computing (CLOUD). Honolulu: IEEE, 2017. 472–479. [doi: 10.1109/CLOUD.2017.67]

[35] Higgins J, Holmes V, Venters C. Autonomous discovery and management in virtual container clusters. The Computer Journal, 2017, 60(2): 240–252. [doi: 10.1093/comjnl/bxw102]

[36] Wan XL, Guan XJ, Wang TJ, Bai GW, Choi BY. Application deployment using microservice and Docker containers: Framework and optimization. Journal of Network and Computer Applications, 2018, 119: 97–109. [doi: 10.1016/j.jnca.2018.07.003]

[37] Martin A, Raponi S, Combe T, Di Pietro R. Docker ecosystem—Vulnerability analysis. Computer Communications, 2018, 122: 30–43. [doi: 10.1016/j.comcom.2018.03.011]

[38] Xu QQ, Jin C, Rasid MFBM, Veeravalli B, Aung KMM. Blockchain-based decentralized content trust for Docker images. Multimedia Tools and Applications, 2018, 77(14): 18223–18248. [doi: 10.1007/s11042-017-5224-6]

[39] Imdoukh M, Ahmad I, Alfailakawi MG. Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 2020, 32(13): 9745–9760. [doi: 10.1007/s00521-019-04507-z]

[40] Oumaziz MA, Falleri JR, Blanc X, Bissyandé TF, Klein J. Handling duplicates in Dockerfiles families: Learning from experts. In: Proc. of the 2019 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Cleveland: IEEE, 2019. 524–535. [doi: 10.1109/ICSME.2019.00086]

[41] Eng K, Hindle A. Revisiting Dockerfiles in open source software over time. In: Proc. of the 18th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Madrid: IEEE, 2021. 449–459. [doi: 10.1109/MSR52588.2021.00057]

[42] Horton E, Parnin C. DockerizeMe: Automatic inference of environment dependencies for python code snippets. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 328–338. [doi: 10.1109/ICSE.2019.00047]

[43] Kitajima S, Sekiguchi A. Latest image recommendation method for automatic base image update in Dockerfile. In: Proc. of the 18th Int'l Conf. on Service-oriented Computing. Dubai: Springer, 2020. 547–562. [doi: 10.1007/978-3-030-65310-1_40]

[44] Wu YW, Zhang Y, Chang JS, Ding B, Wang T, Wang HM. Using configuration semantic features and machine learning algorithms to predict build result in cloud-based container environment. In: Proc. of the 26th IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS). Hong Kong: IEEE, 2020. 248–255. [doi: 10.1109/ICPADS51040.2020.00042]

[45] Henkel J, Bird C, Lahiri SK, Reps T. Learning from, understanding, and supporting devops artifacts for Docker. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Seoul: IEEE, 2020. 38–49. [doi: 10.1145/3377811.3380406]

[46] Henkel J, Silva D, Teixeira L, D'Amorim M, Reps T. Shipwright: A human-in-the-loop system for Dockerfile repair. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 198–199. [doi: 10.1109/ICSE43902.2021.00106]

[47] Piedade B, Dias JP, Correia FF. An empirical study on visual programming Docker compose configurations. In: Proc. of the 23rd ACM/IEEE Int'l Conf. on Model Driven Engineering Languages and Systems: Companion Proc. New York: ACM, 2020. 60. [doi: 10.1145/3417990.3420194]

[48] Ibrahim MH, Sayagh M, Hassan AE. A study of how Docker compose is used to compose multi-component systems. Empirical Software Engineering, 2021, 26(6): 128. [doi: 10.1007/S10664-021-10025-1]

[49] Ksontini E, Kessentini M, Ferreira TDN, Hassan F. Refactorings and technical debt in Docker projects: An empirical study. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 781–791. [doi: 10.1109/ASE51524.2021.9678585]

[50] Lin CY, Nadi S, Khazaei H. A large-scale data set and an empirical study of Docker images hosted on Docker hub. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 371–381. [doi: 10.1109/ICSME46990.2020.00043]

[51] Ibrahim MH, Sayagh M, Hassan AE. Too many images on DockerHub! How different are images for the same system? Empirical Software Engineering, 2020, 25(5): 4250–4281. [doi: 10.1007/s10664-020-09873-0]

[52] D'Urso F, Santoro C, Santoro FF. Wale: A solution to share libraries in Docker containers. Future Generation Computer Systems, 2019, 100: 513–522. [doi: 10.1016/j.future.2019.03.049]

[53] Gkikopoulos P, Schiavoni V, Spillner J. Analysis and improvement of heterogeneous hardware support in Docker images. In: Proc. of the 2021 IFIP Int'l Conf. on Distributed Applications and Interoperable Systems. Valletta: Springer, 2021. 125–142. [doi: 10.1007/978-3-030-78198-9_9]

[54] Anwar A, Mohamed M, Tarasov V, Littley M, Rupprecht L, Cheng Y, Zhao NN, Skourtis D, Warke AS, Ludwig H, Hildebrand D, Butt AR. Improving Docker registry design based on production workload analysis. In: Proc. of the 16th USENIX Conf. on File and Storage Technologies (FAST). Oakland: ACM, 2018. 265–278.

[55] Littley M, Anwar A, Fayyaz H, Fayyaz Z, Tarasov V, Rupprecht L, Skourtis D, Mohamed M, Ludwig H, Cheng Y, Butt AR. Bolt: Towards a scalable Docker registry via hyperconvergence. In: Proc. of the 12th IEEE Int'l Conf. on Cloud Computing (CLOUD). Milan: IEEE, 2019. 358–366. [doi: 10.1109/CLOUD.2019.00065]

[56] Rastogi V, Davidson D, De Carli L, Jha S, McDaniel P. Cimplifier: Automatically debloating containers. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 476–486. [doi: 10.1145/3106237.3106271]

[57] Yin K, Chen W, Zhou JH, Wu GQ, Wei J. STAR: A specialized tagging approach for Docker repositories. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. (APSEC). Nara: IEEE, 2018. 426–435. [doi: 10.1109/APSEC.2018.00057]

[58] Chen W, Zhou JH, Zhu JX, Wu GQ, Wei J. Semi-supervised learning based tag recommendation for Docker repositories. Journal of Computer Science and Technology, 2019, 34(5): 957–971. [doi: 10.1007/s11390-019-1954-4]

[59] Zhao NN, Tarasov V, Albahar H, Anwar A, Rupprecht L, Skourtis D, Paul AK, Chen KR, Butt AR. Large-scale analysis of Docker images and performance implications for container storage systems. IEEE Trans. on Parallel and Distributed Systems, 2021, 32(4):

918–930. [doi: 10.1109/TPDS.2020.3034517]

[60] Li L, Tang T, Chou W. A REST service framework for fine-grained resource management in container-based cloud. In: Proc. of the 8th IEEE Int'l Conf. on Cloud Computing. New York: IEEE, 2015. 645–652. [doi: 10.1109/CLOUD.2015.91]

[61] Huang H, Rao J, Wu S, Jin H, Suo K, Wu X. Adaptive resource views for containers. In: Proc. of the 28th Int'l Symp. on High-performance Parallel and Distributed Computing. Phoenix: ACM, 2019. 243–254. [doi: 10.1145/3307681.3325403]

[62] Mao Y, Oak J, Pompili A, Beer D, Han T, Hu PZ. DRAPS: Dynamic and resource-aware placement scheme for Docker containers in a heterogeneous cluster. In: Proc. of the 36th IEEE Int'l Performance Computing and Communications Conf. (IPCCC). San Diego: IEEE, 2017. 1–8. [doi: 10.1109/PCCC.2017.8280474]

[63] Alzahrani EJ, Tari Z, Lee YC, Alsadie D, Zomaya AY. AdCFS: Adaptive completely fair scheduling policy for containerised workflows systems. In: Proc. of the 16th IEEE Int'l Symp. on Network Computing and Applications (NCA). Cambridge: IEEE, 2017. 1–8. [doi: 10.1109/NCA.2017.8171362]

[64] Menouer T, Cérin C, Leclercq É. New multi-objectives scheduling strategies in Docker SwarmKit. In: Proc. of the 18th Int'l Conf. on Algorithms and Architectures for Parallel Processing. Guangzhou: Springer, 2018. 103–117. [doi: 10.1007/978-3-030-05057-3_8]

[65] Zhang WW, Liu Y, Wang L, Li ZX, Mong Goh RS. Cost-efficient and latency-aware workflow scheduling policy for container-based systems. In: Proc. of the 24th IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS). Singapore: IEEE, 2018. 763–770. [doi: 10.1109/PADSW.2018.8644945]

[66] Mendes S, Simão J, Veiga L. Oversubscribing micro-clouds with energy-aware containers scheduling. In: Proc. of the 34th ACM/SIGAPP Symp. on Applied Computing. Limassol: ACM, 2019. 130–137. [doi: 10.1145/3297280.3297295]

[67] Hamdi N, Chainbi W. A multi-weight strategy for container consolidation. In: Proc. of the 4th IEEE Int'l Conf. on Fog and Edge Computing (ICFEC). Melbourne: IEEE, 2020. 11–18. [doi: 10.1109/ICFEC50348.2020.00009]

[68] Barlaskar E, Kilpatrick P, Spence I, Nikolopoulos DS. Using Docker swarm with a user-centric decision-making framework for cloud application migration. In: Proc. of the 7th Int'l Conf. on Cloud Computing and Services Science. Porto: Springer, 2017. 81–101. [doi: 10.1007/978-3-319-94959-8_5]

[69] Xu B, Wu S, Xiao J, Jin H, Zhang YX, Shi GQ, Lin TY, Rao J, Yi L, Jiang JZ. Sledge: Towards efficient live migration of Docker containers. In: Proc. of the 13th IEEE Int'l Conf. on Cloud Computing (CLOUD). Beijing: IEEE, 2020. 321–328. [doi: 10.1109/CLOUD49709.2020.00052]

[70] Peinl R, Holzschuher F, Pfitzer F. Docker cluster management for the cloud—Survey results and own solution. Journal of Grid Computing, 2016, 14(2): 265–282. [doi: 10.1007/s10723-016-9366-y]

[71] Hoenisch P, Weber I, Schulte S, Zhu LM, Fekete A. Four-fold auto-scaling on a contemporary deployment platform using Docker containers. In: Proc. of the 13th Int'l Conf. on Service-oriented Computing. India: Springer, 2015. 316–323. [doi: 10.1007/978-3-662-48616-0_20]

[72] de Alfonso C, Calatrava A, Moltó G. Container-based virtual elastic clusters. Journal of Systems and Software, 2017, 127: 1–11. [doi: 10.1016/j.jss.2017.01.007]

[73] Rossi F, Nardelli M, Cardellini V. Horizontal and vertical scaling of container-based applications using reinforcement learning. In: Proc. of the 12th IEEE Int'l Conf. on Cloud Computing (CLOUD). Milan: IEEE, 2019. 329–338. [doi: 10.1109/CLOUD.2019.00061]

[74] Brogi A, Neri D, Rinaldi L, Soldani J. Orchestrating incomplete tosca applications with Docker. Science of Computer Programming, 2018, 166: 194–213. [doi: 10.1016/j.scico.2018.07.005]

[75] Mohamed M, Engel R, Warke A, Berman S, Ludwig H. Extensible persistence as a service for containers. Future Generation Computer Systems, 2019, 97: 10–20. [doi: 10.1016/j.future.2018.12.015]

[76] Zhang H, Sun YZ, Xiao L, Tang Y, Hu MM, Du QY, Cai ZB, Feng BM. RainbowD: A heterogeneous cloud-oriented efficient Docker image distribution system. Chinese Journal of Computers, 2020, 43(11): 2067–2083 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02067]

[77] Harter T, Salmon B, Liu R, Arpaci-Dusseau AC, Arpaci-Dusseau RH. Slacker: Fast distribution with lazy Docker containers. In: Proc. of the 14th USENIX Conf. on File and Storage Technologies (FAST). Santa Clara: ACM, 2016. 181–195.

[78] Du L, Wo TY, Yang RY, Hu CM. Cider: A rapid Docker container deployment system through sharing network storage. In: Proc. of the 19th IEEE Int'l Conf. on High Performance Computing and Communications; the 15th IEEE Int'l Conf. on Smart City; the 3rd IEEE Int'l Conf. on Data Science and Systems. Bangkok: IEEE, 2018. 332–339. [doi: 10.1109/HPCC-SmartCity-DSS.2017.44]

[79] Gu L, Tang QZ, Wu S, Jin H, Zhang YX, Shi GQ, Lin TY, Rao J. N-docker: A NVM-HDD hybrid Docker storage framework to improve Docker performance. In: Proc. of the 16th IFIP Int'l Conf. on Network and Parallel Computing. Hohhot: Springer, 2019. 182–194. [doi: 10.1007/978-3-030-30709-7_15]

[80] Ahmed A, Pierre G. Docker image sharing in distributed fog infrastructures. In: Proc. of the 2019 IEEE Int'l Conf. on Cloud Computing Technology and Science (CloudCom). Sydney: IEEE, 2019. 135–142. [doi: 10.1109/CloudCom.2019.00030]

[81] Nguyen TL, Nou R, Lebre A. YOLO: Speeding up VM and Docker boot time by reducing I/O operations. In: Proc. of the 25th Int'l Conf. on Euro-Par 2019: Parallel Processing. Göttingen: Springer, 2019. 273–287. [doi: 10.1007/978-3-030-29400-7_20]

[82] 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 with English abstract). http://www.jos.org.cn/1000-9825/5816.htm [doi: 10.13328/j.cnki.jos.005816]

[83] Paraiso F, Challita S, Al-Dhuraibi Y, Merle P. Model-driven management of Docker containers. In: Proc. of the 9th IEEE Int'l Conf. on Cloud Computing (CLOUD). San Francisco: IEEE, 2016. 718–725. [doi: 10.1109/CLOUD.2016.0100]

[84] Jha DN, Garg S, Jayaraman PP, Buyya R, Li Z, Ranjan R. A holistic evaluation of Docker containers for interfering microservices. In: Proc. of the 2018 IEEE Int'l Conf. on Services Computing (SCC). San Francisco: IEEE, 2018. 33–40. [doi: 10.1109/SCC.2018.00012]

[85] Zerouali A, Mens T, Robles G, Gonzalez-Barahona JM. On the relation between outdated Docker containers, severity vulnerabilities, and bugs. In: Proc. of the 26th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Hangzhou: IEEE, 2019. 491–501. [doi: 10.1109/SANER.2019.8668013]

[86] Zerouali A, Mens T, Decan A, Gonzalez-Barahona J, Robles G. A multi-dimensional analysis of technical lag in debian-based Docker images. Empirical Software Engineering, 2021, 26(2): 19. [doi: 10.1007/s10664-020-09908-6]

[87] Zerouali A, Mens T, De Roover C. On the usage of JavaScript, Python and Ruby packages in Docker hub images. Science of Computer Programming, 2021, 207: 102653. [doi: 10.1016/j.scico.2021.102653]

[88] Liu PY, Ji SL, Fu LR, Lu KJ, Zhang XH, Lee WH, Lu T, Chen WZ, Beyah R. Understanding the security risks of Docker hub. In: Proc. of the 25th European Symp. on Research in Computer Security on Computer Security. Guildford: Springer, 2020. 257–276. [doi: 10.1007/978-3-030-58951-6_13]

[89] Zhang MW, Marino D, Efstathopoulos P. Harbormaster: Policy enforcement for containers. In: Proc. of the 7th IEEE Int'l Conf. on Cloud Computing Technology and Science (CloudCom). Vancouver: IEEE, 2015. 355–362. [doi: 10.1109/CloudCom.2015.96]

[90] Tak B, Kim H, Suneja S, Isci C, Kudva P. Security analysis of container images using cloud analytics framework. In: Proc. of the 25th Int'l Conf. on Web Services. Seattle: Springer, 2018. 116–133. [doi: 10.1007/978-3-319-94289-6_8]

[91] Fernandez GP, Brito A. Secure container orchestration in the cloud: Policies and implementation. In: Proc. of the 34th ACM/SIGAPP Symp. on Applied Computing. Limassol: ACM, 2019. 138–145. [doi: 10.1145/3297280.3297296]

[92] Gantikow H, Reich C, Knahl M, Clarke N. Rule-based security monitoring of containerized environments. In: Proc. of the 9th Int'l Conf. on Cloud Computing and Services Science. Heraklion: Springer, 2019. 66–86. [doi: 10.1007/978-3-030-49432-2_4]

[93] Simonsson J, Zhang L, Morin B, Baudry B, Monperrus M. Observability and chaos engineering on system calls for containerized applications in Docker. Future Generation Computer Systems, 2021, 122: 117–129. [doi: 10.1016/j.future.2021.04.001]

[94] Ruan BW, Huang H, Wu S, Jin H. A performance study of containers in cloud environment. In: Proc. of the 10th Asia-Pacific Services Computing Conf. on Advances in Services Computing. Zhangjiajie: Springer, 2016. 343–356. [doi: 10.1007/978-3-319-49178-3_27]

[95] Varghese B, Subba LT, Thai L, Barker A. DocLite: A Docker-based lightweight cloud benchmarking tool. In: Proc. of the 16th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing. Cartagena: IEEE, 2016. 213–222. [doi: 10.1109/CCGrid.2016.14]

[96] Santos EA, McLean C, Solinas C, Hindle A. How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers. Journal of Systems and Software, 2018, 146: 14–25. [doi: 10.1016/j.jss.2018.07.077]

[97] Garg SK, Lakshmi J, Johny J. Migrating VM workloads to containers: Issues and challenges. In: Proc. of the 11th IEEE Int'l Conf. on Cloud Computing (CLOUD). San Francisco: IEEE, 2018. 778–785. [doi: 10.1109/CLOUD.2018.00106]

[98] Yoshimura T, Nakazawa R, Chiba T. ImageJockey: A framework for container performance engineering. In: Proc. of the13th IEEE Int'l Conf. on Cloud Computing (CLOUD). Beijing: IEEE, 2020. 238–247. [doi: 10.1109/CLOUD49709.2020.00043]

[99] Affetti L, Bresciani G, Guinea S. aDock: A cloud infrastructure experimentation environment based on open stack and Docker. In: Proc. of the 8th IEEE Int'l Conf. on Cloud Computing (CLOUD). New York: IEEE, 2015. 203–210. [doi: 10.1109/CLOUD.2015.36]

[100] Zhang R, Zhong AM, Dong B, Tian F, Li R. Container-vm-pm architecture: A novel architecture for Docker container placement. In: Proc. of the 11th Int'l Conf. Held as Part of the Services Conf. Federation on Cloud Computing. Seattle: Springer, 2018. 128–140. [doi: 10.1007/978-3-319-94295-7_9]

[101] Haque MU, Iwaya LH, Babar MA. Challenges in Docker development: A large-scale study using stack overflow. In: Proc. of the 14th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM). Bari: ACM, 2020. 7. [doi: 10.1145/3382494.3410693]

[102] Gholami S, Goli A, Bezemer CP, Khazaei H. A framework for satisfying the performance requirements of containerized software

systems through multi-versioning. In: Proc. of the 2020 ACM/SPEC Int'l Conf. on Performance Engineering. Edmonton: ACM, 2020. 150–160. [doi: 10.1145/3358960.3379125]

附中文参考文献:

[76] 张浩, 孙毓忠, 肖立, 唐勇, 胡满满, 杜沁园, 蔡志彬, 冯百明. RainbowD: 一种异构云环境下高效的Docker镜像分发系统. 计算机学报, 2020, 43(11): 2067–2083. [doi: 10.11897/SP.J.1016.2020.02067]

[82] 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 [doi: 10.13328/j.cnki.jos.005816]

吴逸文(1996－), 女, 博士生, CCF 学生会员, 主要研究领域为实证软件工程, 容器运维, 数据挖掘.

王涛(1984－), 男, 博士, 副研究员, 主要研究领域为软件工程, 机器学习, 数据挖掘.

张洋(1991－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为实证软件工程, 开发运维一体化.

王怀民(1962－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为软件工程, 分布式计算, 云际计算.