**大 连 理 工 大 学 本 科 外 文 翻 译**

**基于容器技术的云的性能比较**

**A performance comparison of container-based technologies for the Cloud**

学 部（院）： 软件学院

专 业： 软件工程

学 生 姓 名：

学 号：

指 导 教 师：

完 成 日 期： 2016年12月25日

大连理工大学

Dalian University of Technology

**摘要：**通过虚拟化技术，云计算可以实现高效和可扩展的方式来使用服务器。在IaaS云模型中，许多虚拟化服务器（实例）可以在单个物理机上创建。现在有很多云提供商提供这种被广泛使用的能力。然而，云计算也有开销并且限制可扩展性和灵活性，特别是当不同的用户使用云资源有不同的需求时。为适应这种环境，被广泛采用的云计算和虚拟化的代替品是微主机服务和基于容器的解决方案。基于容器的技术如Docker微服务云架构，使用捆绑应用程序和数据的方式使它们易于部署并且方便使用。 Docker只是许多已提出的这样的解决方案之一。本文的目的是比较和对比一系列现有的基于容器的云技术，并评估其利弊和整体表现。基于OpenStack的澳大利亚全国电子研究协作工具和资源（NeCTAR）研究云（www.nectar.org.au）被用于此目的。我们描述实验的设计和基准，并将这些与文献综述结果相关联。

1. 介绍：

现在，市场对云平台和相关的虚拟化技术有很大的需求。许多软件公司例如VMware（VMware），Citrix（Xen），Microsoft（Hyper-V）等都用自己的解决方案主导着虚拟化市场，并且那些面向英特尔和AMD的硬件公司现在也提供先进的处理器来支持虚拟化。这些技术总体上用于服务器整合，通常应用在数据中心中，通过弹性缩放以灵活的方式为外部社区提供大量服务器集合。现在有很多关于虚拟化性能的研究，有些工作集中在HPC设施[1,2]，也有一些人则专注于云环境。以前的研究表明基于管理程序的虚拟化技术有很高的性能开销。此外，他们受I / O限制，因此通常避免HPC环境。最近基于容器的虚拟化技术和对微宿主服务的支持已经被很大程度上接受了，因为它提供了一个轻量级的解决方案，允许以更简单和以性能为导向的方式捆绑应用程序和数据，也使它可以运行在不同的云基础设施上。这种处理虚拟化的方式提供了扩展性很好的可部署系统，而且没有传统管理程序的高性能带来的困扰以及管理大规模云基础设施的开销[3]。在这项文章中，我们对微宿主服务进行了回顾，并进行了一些实验，为基于容器的虚拟化技术提供了一个全面的性能评估。我们特别关注一些具有代表性的系统如Docker [4-6]和Flockport（LXC）[7,8]等作为主要对象。

这项工作的目的是比较基于容器的虚拟化技术在云上的性能。这项工作主要关注CPU，内存以及I / O设备能力。为了满足这些标准，定义了四个目标：

~严格审查相关工作的性能实验，以测量现有的不同虚拟化技术在不同环境下的性能;

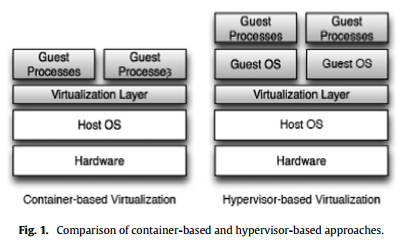
~确定面向绩效的案例研究，来评估虚拟化技术在云上的性能;

~实施几个案例来评估微宿主技术性能;

~比较从实验中获得的结果，并确定在这些实验条件下的微主机技术的优缺点。

本文组织如下：第2节概述了各种基准化应用程序的虚拟化技术和相关工作。更准确地说，它描述了基于容器的虚拟化和基于管理程序的虚拟化以及这些解决方案的代表性示例，包括Docker，LXC（Flockport）和CoreOs Rocket。第3节比较了可能影响其性能的基于容器的技术的关键特性。第4节中描述了执行以评估虚拟化性能的实验设计，包括用于执行基准测试案例研究的云环境，系统架构和基准测试工具。实验的实施细节和性能比较的结果在第4节中给出。性能比较的摘要和进一步研究的领域在第5节中给出。

# 2.背景和相关工作

****

**Fig.1** 基于容器和基于应用程序方法的比较

资源的虚拟化通常包括利用宿主机操作系统上方的附加软件层来处理资源。 这样的虚拟机（VM）可以被认为是单独的执行环境。几种用于虚拟化目的方法[9]。 还有一种流行的技术是基于管理程序的虚拟化。知名的基于管理程序的虚拟化的解决方案是：KVM和VMware。 为了使用这种技术，应该在底层物理系统上方有一个虚拟机监视器。每个虚拟机还支持（隔离的）客户操作系统。 在这种虚拟化方法中，一个主机操作系统很可能支持许多客户操作系统[9]。

基于容器的虚拟化技术代表了另一种方法。 在这种模型中，硬件资源通过具有（安全）相互隔离属性的许多实例来划分。这两种技术的区别可以在图1中看到。这里，客户进程使用基于容器的技术直接获得抽象，因为它们通过虚拟化层可以直接在操作系统（OS）级别上操作。然而，在基于管理程序的方法中，每个客户操作系统通常有一个虚拟机[9]。 在基于容器的解决方案中，一个操作系统内核通常在虚拟实例之间共享。因此，有一种设想是基于容器方法的安全性比基于管理程序的弱。从用户的角度来看，容器作为独立的操作系统运行，它们可以独立于硬件和软件运行[10]。

Biederman [11]认为内核命名空间负责处理容器的隔离。这被认为是Linux内核特殊属性，它允许进程获得必要的抽象层次。尽管事实上容器和命名空间层之间没有交互，但在主机操作系统和客户进程之间仍存在隔离，每个容器都有自己的操作系统。根据Biederman [11]，文件系统，进程标识符，网络以及进程间通信是通过命名空间进行隔离的。然而，基于容器的虚拟化技术中的进程组存在资源利用的限制。此过程由cgroups管理[12]。确切地说，cgroup负责确定CPU，内存以及容器虚拟化中的I / O利用率的优先级。然而，使用容器的某些技术可以根据cgroup的一致性来实现它们对资源的管理。

使用这种基于容器的解决方案允许在绑定的主机环境中动态部署和使用微服务。微服务模式在软件架构中不是一个新的想法。如今，它们被广泛认为是开发应用程序的有效解决方案。在微服务架构之前，服务开发的一般方法是创建庞大的应用程序。从功能的角度来看，这需要一个单一的环境来处理所有的事情。在云环境中，这些问题可以通过IaaS， PaaS和SaaS的方法来克服。然而当有许多特定项目和社区带着自身的多样化软件和数据需求参与进来时，这样的解决方案又受到了挑战。在这种环境下，轻量级的支持云的解决方案是有益的。微服务就是这样一种模式。

微服务架构的主要概念是“分而治之”。 基本上，微服务用小的敏捷可控的多个小规模代码基础替代单个更大的代码库。 这些代码彼此间具有单一的API关系。 这个概念背后的好处是每个群体都可以单独工作，并彼此保护/断开连接 —也就是所谓的无关圈。 然而，这些无关圈可以在某些情况下连接，例如当服务之间存在依赖关系时。

现在存在一系列基于容器的微主机服务。 其中最成熟的是Docker，CoreOS和LXC。Docker提供了一种不太复杂的方法来将应用程序包装到包含其执行所需环境的容器中。该方法通过一种集合工具和具有内核级结构的集成应用编程接口引导技术来执行。 Linux容器，控制组以及写入时复制文件系统。充当容器的文件系统Docker依赖于高级多层统一文件系统（AuFS）。 AuFS能够显式地叠加单个或多个可用的文件系统。它使Docker能够基于其他镜像创建容器。例如，一个人可以使用Ubuntu镜像，它又可以为多个其他容器提供基础。在高级分层统一文件系统的帮助下，Docker利用了Ubuntu的单层。这极大地节省了存储空间，并减少了内存的使用，使容器可以快速启动。 AuFS有一个更明显的优点，使其能够创建镜像版本。每个最新版本都标记为diff，这个优势可以标识两个文件之间的差异。这使镜像文件的体积减小。 AuFS的另一个优点是，可以跟踪每个镜像版本的每个修改 - 就像软件开发代码版本控制系统[4]。

CoreOS是相对较新的Linux分行版本，其设计目的是提供操作软件系统堆栈所需的特性。 这种技术提供了一个减缩版的Linux内核，最大限度地减少开销。 此外，CoreOS支持利用工具集群，以确保冗余并且保护系统免于故障。

最近，CoreOS推出了一个新的产品--rocket容器运行时（rkt）[14]，它按照规范运行应用容器。 此技术的主要目的是为容器构建指定的模型。 此方法还支持Amazon Machine Images。 rocket容器运行时是Docker的一个选项，它包括高级安全策略以及在服务器上运行所需的其他需求。 rocket容器运行时与应用容器规范相对应，应用容器规范为容器提供了一组新的格式，使其容易携带或移动。 在Docker中，每个进程都通过守护进程运行，从安全角度来看，它不能提供与Rocket一样多的保证。 为了纠正这种现象，已经建议Docker应该被完全重写。

LXC在标准Linux内核中，它使设备能够管理容器和操作系统镜像。容器可以被认为是与主机操作系统一起执行的轻量级OS。 容器不模仿硬件层，因此它们可以在没有其它性能开销的情况下以几乎本机同速度执行。 在它们的标准使用方法中，建立应用以及网络栈，并且以特定形式在裸机服务器上进行测试。例如，PHP和MySQL，Nginx的和Drupal可以安装并配置同时运行。然而，当前应用程序专用于已经建立并且不能简单地迁移的机器。 虚拟机可以被安装和迁移，并且它可以给予一些容易移动的感觉，但是这损害了性能。

LXC容器基本上可以提供裸机吞吐量并能够将堆栈建立到容器中来简单地提供在系统中迁移的能力。LXC容器具有的性能和灵活性，让人存在它是一个单独的服务器的错觉。可以克隆，备份和快照容器。 LXC使容器管理更加便捷，并在执行和启动应用程序方面带来了灵活性。 Flockport确保web堆栈以及LXC容器中的软件可以在任何基于Linux的机器上启动。 LXC容器直接支持灵活性和吞吐量，而Flockport是发布LXC容器并简化其使用的工具。

一些研究人员集中精力在减少虚拟化技术和非虚拟化方法之间性能和优化方面的差异。 用到的方法和工具在这些研究中通常是不同的。 此外，检查和比较的技术组合物也不同。 例如，在最近的研究论文中检查和比较本地系统和它们的可视化版本之间的差距以及基于容器和基于管理程序的虚拟化技术之间的性能差异。我们注意到这是一个快速发展的领域，因此一些早期的论文是基于已经过时的软件。而且他们不对最近的可视化技术进行分析。

Hwang等人比较了四种不同的基于管理程序的虚拟化技术。 在[15]中。 他们没有发现任何有显著高性能的管理程序。 因此，他们提出的建议是在云设施中使用各种软件和硬件平台，以满足客户的要求。

Abdellatief et(al。 [16])在不同的场景中对VMware，Microsoft Hyper-V和Citrix Xen等技术进行了性能比较。 他们使用的评估方法是为应用定制的SQL实例。 在这种方法的帮助下，他们模拟了数百万的产品，客户和订单。 Varrette等人进行了同样的分析。 [17]，但使用的是不同的技术和相关的测试环境。 他们使用基于系统内核的虚拟机代替Microsoft Hyper-V，并进行了与高性能计算相关的实验。 他们的实验集中在功率消耗，能量效率以及可扩展性。

尽管事实上存在虚拟化开销不一致的演示，[17]但可以确定几乎每个虚拟化管理程序的虚拟化层对虚拟化环境的性能，特别是对于高性能计算域提供了显着的影响。

最近的出版物考虑了管理程序与容器方法之间的相似性或不相似性。根据Dua et al [18]，容器在PaaS设施中变得流行起来。 两种类型的虚拟化（即KVM，Xen和LXC）的代表由Estradam等人进行了基准测试[19]。他们研究的主要方法是测量每个提到的技术的相似性以及运行时性能的差异。他们的实验和基准测试的基础是支持基于序列的应用程序。

Felter et al[5]还分别比较了管理程序和基于容器系统的技术，即KVM和Docker。他们对CPU，内存，存储以及网络带宽和延迟进行了全面的分析。根据其基准测试结果，Docker容器的性能几乎与“裸机”系统相同。 他们的基准是基于内存传输，浮点处理，网络资源，块I / O以及数据库容量。 然而，作者的工作没有系统地检查容器对传统管理程序的影响。

Heroku PaaS提供商[20]中引入了利用容器以高效和可重复的方式部署应用程序。Heroku提供了一个容器作为一个具有额外隔离属性的进程，而不是将其视为一个虚拟服务器。因此，应用程序部署容器提供了一种轻量级技术，其中的虚拟开销和虚拟机几乎相同。 它还具有作为标准进程的资源共享属性。 这种容器在Google的基础设施中被大量使用。 此外，Docker提供了图像的标准格式以及应用程序容器的管理工具。

与以前的出版物相比，这项工作的主要特点是，本工作中的分析具体涉及基于开源容器技术的性能基准，从而来确定相关的优点和缺点。

# 3. 微托管的比较

每个基于容器的技术都有自己的特性。 本节介绍了当前领先的基于容器的技术的一些关键特性，这些技术可能会对性能产生影响。 我们注意到，即使CoreOs Rocket在上一节中已经确定，但由于CoreOS尚未发布其产品的官方版本，因此未对Rocket进行性能评估。

## 3.1.Docker

Docker利用了几个Linux内核功能，以便以独立的方式运行容器[9]。

• **namespaces**: Docker使用命名空间来部署容器。 Docker使用了几种类型的命名空间来执行创建隔离容器的任务[4]：

- Docker使用pid作为容器的基础，确保容器中的所有进程不会影响其他容器中的进程;

- 它使用net以便管理网络接口，或者更精确地说，它提供关于网络的系统资源的隔离;

- ipc用于提供对特定进程间通信（IPC）资源的隔离，即系统V IPC对象和POSIX消息队列。 这意味着每个IPC命名空间都有自己的进程间通信资源。

- 为了允许进程拥有自己的文件系统视图和挂载点，Docker使用mnt命名空间。

- 通过uts执行内核和版本标识符的隔离。

•控制组：Docker使用cgroup，以便现有容器可以共享可用的硬件资源，并且可能在给定时间限制这些资源。

•联合文件系统是一种通过建立用于提供容器构建块的层来运行的文件系统。

•容器格式被视为集成了所有前述机制的包装器。

## 3.2 LXC

Linux Container是一种基于容器的虚拟化技术，使用通用和灵活的API实现，可轻松构建轻型Linux容器[21]。 另一方面，Docker是一种基于容器的应用程序技术。 他们有一些共同的特点，但也有很多不同之处。 首先，LXC是一种操作系统级虚拟化技术，用于在单个LXC主机上执行几个隔离的Linux容器。 它不使用虚拟机，而是允许利用具有其自己的CPU，存储器，阻塞I / O，网络以及资源控制机制的虚拟环境。 这通过LXC主机上的Linux内核中的命名空间和cgroups功能提供。 它类似于chroot，但提供了更多的隔离。 LXC支持各种虚拟网络类型和设备。其次，Docker利用固定的层来实现良好的结构重用，但是这可能以复杂性和吞吐量为代价。 每个容器的一个应用的限制降低了利用可能性。 使用LXC，可以创建单个和/或多个应用程序。 此外，LXC允许创建多个系统容器，其可以是利用btrfs文件运行的仅一个子卷的克隆。LXC的这一特性可以用来解决文件系统级别的复杂问题。 第三，LXC提供了设计和执行容器设备和权限的通用列表。 最后，LXC允许创建无特权的容器，确保非root用户可以构建容器。 Docker还不支持此功能。

## 3.3. CoreOS rocket

Rocket [14]是CoreOS引入的基于容器的技术，它提供了Docker的替代方案。 Rocket和Docker都以虚拟容器的形式引入了应用程序部署的自动化，这些虚拟容器可以基于服务器的特性独立执行。然而，Docker已经发展成为一个复杂的环境，支持多样化的需求和操作，而Rocket被构造为执行简单的功能，但以安全的方式部署应用程序。 Rocket被用作执行描述图像设计的应用程序容器的命令行工具。Rocket专注于CoreOS引入的应用程序容器规范，作为允许容器容易迁移的描述的组合。正如Polvi [14]所认识到的，Rocket可能比Docker更难使用，因为Docker通过描述性接口简化了构建容器的整个过程。 [14]认为Rocket应该作为一个基于命令行的环境，并且不太容易改变。

# 4. 评估方法和基准

有很多角度可以用来比较技术，特别是从性能的角度。 为了从其开销的角度评估容器技术，有必要了解（测量）基于非虚拟化环境产生的开销。这里进行的分析集中在一系列性能标准：CPU，内存，网络带宽延迟和存储开销的性能。 所有的基准都多次实验重复15次，以评估各种结果的准确性和一致性。 记录平均时间和标准偏差。

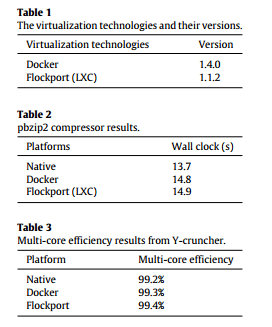
这些工作的云环境是澳大利亚全国电子研究协作工具和资源（NeCTAR）研究云（www.nectar.org.au）。 NeCTAR为澳大利亚的所有研究人员提供了一个云环境。 它在八个可用区域提供30,000台服务器，通常位于州首府（墨尔本，堪培拉，霍巴特等）。 NeCTAR项目由墨尔本大学领导，由教育部资助。 NeCTAR利用OpenStack中间件来实现云基础架构。

性能研究都在NeCTAR研究云的实例上执行。以下实例配置用于实验：

型号：处理器：AMD Opteron 62xx类@ 2.60 GHz; 处理器ID：AuthenticAMD系21型号1步进2; 内存：3955 MB; 操作系统：Ubuntu 12.04（64位）。

虚拟化技术及其版本见表1。

为了一致，所有Docker和Flockport（LXC）容器都使用了Ubuntu 64位系统镜像。 此外，它们都是在基于Ubuntu 12.04 64位的主机操作系统上运行的。



**Table 1** 虚拟化技术和它们的版本

**Table 2** pbzip2 压缩结果

**Table 3** 从Y-cruncher得到的多核执行效率结果

在本节中说明了进行的实验的结果。如前所述，基准测试工具评估了CPU，内存，网络带宽和延迟，存储开销性能。

## 4.1. CPU 性能

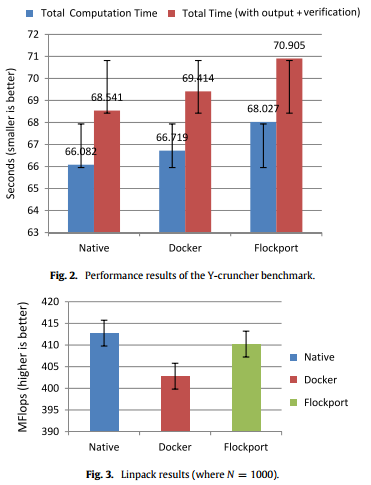
评估CPU性能的第一种情况是使用压缩程序。压缩是云环境处理的常用模块。 PBZIP2 [22]是bzip2块排序数据压缩程序的并行实现。它利用p个线程并且可以达到几乎线性的加速度。 使用pbzip2-1.1.12版本来压缩文件。输入文件（enwik8）[23]，它是100 MB转储数据，并经常用于压缩测试目的。为了集中压缩，使用900kB BWT块大小和900kB文件块大小。

pbzip2压缩器的性能如表2所示。从CPU评估的角度看，Docker的性能略胜LXC。 在Flockport的情况下，平均经过时间为14.9s，标准偏差为±0.03s，而Docker的平均时间为14.8s，标准偏差为±0.01s。 但是，应该考虑输入文件的大小。 如果大小增加，结果的差异可能很大。

第二个CPU基准测试工具是Y-cruncher [24]，用于计算Pi。 它通常作为CPU的一个压力测试工具执行，并经常用作在多核系统中多线程工具的测试。 除了计算Pi的值，Y-cruncher还可以计算其他常数的范围。 Y-cruncher计算各种结果，例如多核效率，计算时间和总执行时间。使用总时间来确认结果，并且其是包括请求执行结果时间的总计算时间。

Y-cruncher基准测试工具的性能结果可以在图2中看到.在计算时间方面，Docker的性能类似于本地（非虚拟化）系统，而Flockport平均需要大约2秒。 这些系统的多核效率结果如表3所示。该评估描述了如何有效地利用CPU计算Pi。这个模式还表明Docker的性能比Flockport稍好。

下一个基准测试工具是标准的HPC基准测试工具：Linpack。此工具有两个选项。第一个是Intel [25]的优化版本，第二个[26]允许在所有机器上运行，而不仅仅是Intel机器。Linpack找到了一个对应系统的解决方案，利用一种方法，执行数值分析的“下部”分解与部分枢转。大量的计算操作包括将标量的过程与双精度浮点格式的向量相乘以及将结果添加到不同向量的过程。基准测试工具基于所选计算机体系结构的线性代数函数。主Linpack函数利用大小为N的随机矩阵M和被确定为M \* X = V的向量V.Linpack基准测试工具执行两个步骤，如下：M的“下上”分解，随后是“下上'分解用于解决线性问题M \* X = V。Linpack的结果通常以每秒MegaFLOPS浮点运算的形式提供。执行此工具并增加N的值，从实验可以看出，CPU使用率的变化和各种阶段可以被识别：上升区，本地存储器或处理器中没有压力;平坦区域，压力产生在处理器功能中;以及衰减区域，在本地高速缓存存储器中产生压力。

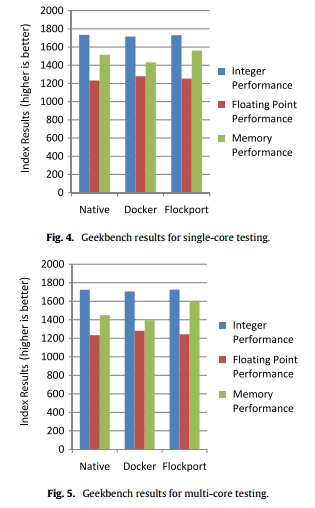


**Fig.2** 参照Y-cruncher的表现结果

**Fig.3** Linpack 结果（当N等于1000时）

Linpack在微主机环境中运行的结果可以在图3中看到。 这些结果是通过应用N = 1000的特定场景获得的。如图所示，即使它们之间的差距相对较小，Flockport在执行Linpack基准时的性能略好于Docker。

用于评估CPU性能的最终工具是Geekbench [27]。 此工具可用于测试浮点单元的性能以及内存系统的吞吐量。当前工作评估了吞吐量特征的上限。与Ycruncher相比，Geekbench支持单核和多核架构的评估。它可以生成多个索引的各种工作负载，如整数性能，浮点性能和内存性能。 此外，可以生成完整系统的索引。

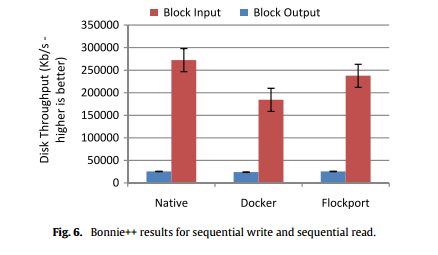


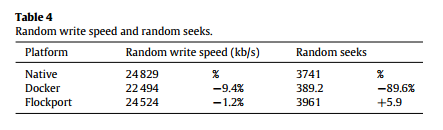
**Fig.4** 单核测试的Geekbench结果

**Fig.5** 多核测试的Geekbench结果

从图4和5可以看出，在单核或多核测试结果上没有显著差异。然而，关于内存性能，Flockport分别比单核和多核测试的Docker性能产生大约100点和200点得优势。

## 4.2. Disk I/O 性能





**Fig.6** Bonnie++的队列读和队列写的结果

**Table 4** 自由写速度和自由查找

性能的一个关键方面是评估磁盘I/O性能，特别是作为非临时存储的卷附加到实例。这些卷是在与相关实例相同的可用区中创建的。用于评估微主机环境的第一个应用基准工具是Bonnie ++ [28]。 Bonnie ++是一个描述磁盘吞吐量的开源应用程序。 Bonnie ++必须针对不同的情况进行配置，因此使用了一个公共测试文件。为此目的使用了一个4 Gb数据集。图6分别给出了基于顺序写入和读取的块输出和块输入的Bonnie ++结果。 所有三个系统在顺序写入方面产生几乎相同的结果。 然而，与Docker相比，Flockport在顺序读取文件方面表现出稍好的性能。

通过Bonnie ++基准测试应用程序，可以对文件中进行读取，写入和刷新速度评估。 表4显示了每个系统的随机写入速度以及随机查询。

在随机写入速度结果之间存在相对校准，在该结果处读取文件，然后将其写入并刷新到磁盘，如图1所示。 而且，系统的设置也导致在随机搜索评估中结果是不同的，其显示出了Bonnie ++可以每秒寻求的块的数量。 在这种情况下，Flockport比Docker的结果好100％，比本地平台好6％。

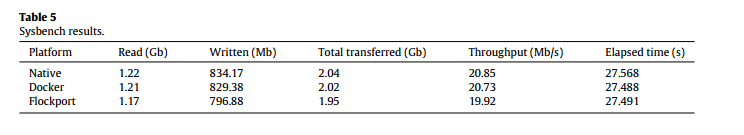
为了进一步评估磁盘I/O吞吐量，使用Sysbench基准测试工具[29]。 这个工具涉及许多模块作为设计基础。它还可以用作跨平台测量操作系统特性的多线程测量软件。Sysbench的主要结果是在不部署数据库系统的情况下以高速获得关于系统吞吐量的表示。该软件的发布版本允许评估系统特性，例如文件I/O吞吐量，内存分配和传输速度调度程序吞吐量。在执行当前场景中，该工具用于评估平台读取以及向指定文件写入的性能。 为了评估文件IO性能，应该创建一个大小比可用RAM大的测试文件。 这里使用的测试文件的大小为35GB，分为128个文件，每个文件大小为270Mb。

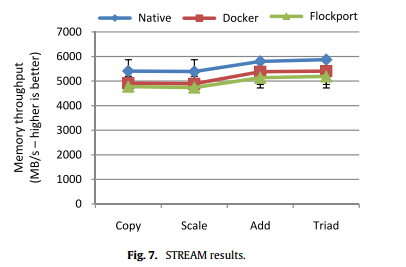
本次测量结果在表5中给出。在Docker和Flockport上任务执行所花费的总时间为27.488秒，标准偏差±0.0003秒和27.491秒，标准偏差±0.0001秒。没有显着性差异。

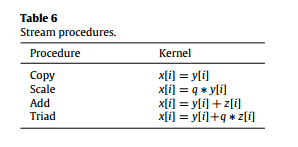
Sysbench得到的结果也反映了通过Bonnie++获得的结果。在本机以及Flockport显示大致相同的结果，没有任何可辨别的差异。然而，Docker在一些地方的表现优于另外两个。

为了测试更彻底，应该注意到Native和Flockport的磁盘镜像使用的是默认文件格式，而Docker应用了高级多层统一文件系统来确保分层和映像版本。 因此，在该实验中应该进行其它测试。

## 4.3. Memory 性能

本节介绍了内存I/O性能的评估。用于测试微主机环境的基准工具是STREAM软件[30]。STREAM使用直接的矢量内核进程来评估内存吞吐量。这个工具生成四个过程的结果，即复制，缩放，添加以及三元组。这些程序执行结果及其如何计算如表6所示。  






**Table 5** Sysbench 结果

**Fig.7** STREAM 结果

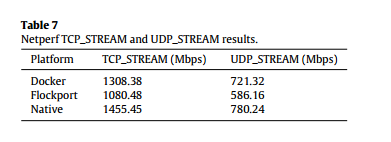
**Table 6** Stream 程序

根据STREAM软件估算的吞吐量和CPU高速缓存的大小之间存在强关联。此外，规则是每个流阵列必须至少是可访问的高速缓存存储器大小的四倍。 因此，必须正确地建立软件中的流阵列的大小。 如图7所示，结果之间的区别不是很大，但是Docker产生的结果略好于Flockport，并且与本地平台几乎相同。

## 4.4. Network I/O 性能

本小节描述了容器环境下其网络I / O的性能。 在同一主机上运行的任何容器，更确切地说，在同一主机网桥上，可以通过IP彼此联系。网络地址转换（NAT）网络原理被用于所有业务。这使得所有容器可以在不同主机上与外部（包括其他Docker容器）联系，但是它不允许外部网络与容器通信。可以通过将容器端口映射到主机网络接口上的端口来避免此规定。Linux容器具有相同的网络。 测试用例在两个不同主机上的两个Docker容器上进行。 相同的条件也适用于Flockport容器。 在这两个一个作为服务器和另一个作为客户端。在本地条件下，使用两个相同的NeCTAR研究云实例。

Netperf基准测试工具[31]用于测量网络I / O。此工具有许多预定的子测试来评估服务器和客户端之间的网络吞吐量。它允许使用TCP或UDP协议在单个方向进行数据传输。 在netperf客户端和netserver之间建立连接所花费的时间不包含在这些评估中。评估结果在表7中示出。



**Table 7** Netperf 的TCP\_STREAM和UDP\_STREAM 结果

表7显示了Docker和Flockport分别以TCP\_STREAM和UDP\_STREAM方式得到的结果。对于TCP\_STREAM测试用例，Docker比Flockport多大约200 Mbps，而对于UDP\_STREAM测试用例，Docker再次领先了大约150 Mbps的性能。 此外，如上所述，Netperf基准测试工具具有包括请求和响应的评估，通过其测量TCP和UDP事务量。在以下情况下会建立连接：netperf客户端将请求分派给netserver，netperf服务器向netperf客户端分派响应。

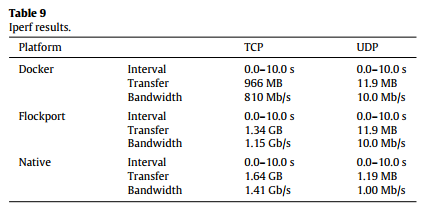
如表8所示，Flockport再次产生对于TCP\_RR以及UDP\_RR测试情况的最差结果。

为了全面了解网络I/O性能，Iperf套件[32]也被用来测试网络吞吐量：。 Iperf提供了一个完整的套件用于使用TCP或UDP协议评估连接性能。 表9显示了TCP和UDP流量的Iperf测试的结果。

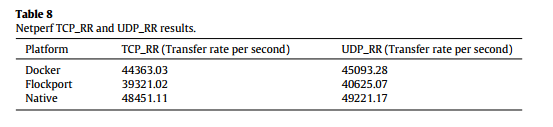
从表9可以看出，与之前的I / O性能测试案例相比，Docker更慢。可能是因为TCP窗口大小和UDP缓冲区大小造成的。也就是说，在没有验证的情况下，给定连接的时间窗口中缓冲的数据量不同。数据的大小可以在2到65,535字节的范围之间，但是它们在Iperf中是小的（60.0kB）并且未被调整。

# 5. 结论和未来要做的工作

基于容器的技术是使用虚拟机管理程序作为云的基础。现代的基于容器的方法被认为是轻量级的。在本文中，对领先的微主机虚拟化方法进行了全面的性能评估，特别关注Docker和Flockport以及它们与本地平台的比较。 如上所述，CoreOS没有被考虑，因为在撰写本文时系统不可用。



**Table 9** Iperf 结果



**Table 8** Netperf TCP\_RR和UDP\_RR结果

关于所进行的实验结果，可以看到许多常见的模式。如图所示，Docker或Flockport对内存利用率或CPU几乎没有开销，而I / O和操作系统交互会产生一些开销。这些情况下的开销是用于每个输入 - 输出操作的附加周期导致的。 因此，与具有较低输入 - 输出需求的应用相比，具有很多输入 - 输出操作的应用有更多缺点。输入输出等待时间导致开销加大。程序实时工作所需的CPU周期也可能导致性能下降。

Docker包括一些其他功能，如网络地址转换，这有助于减少使用Docker容器的一些困难。然而，这些能力会接影响吞吐量和质量的输入。 因此，在某些情况下，与Flockport相比，即使不使用额外功能的Docker容器也不会更快。 文件系统或磁盘请求密集型软件必须通过应用卷来使用高级多层统一文件系统。 网络地址转换的影响可以通过使用nethost来移除。 然而，这抵消了网络命名空间的优点。 最后，Kubernetes [33]建议的每个Docker容器的一个IP地址的设计可以支持质量保证以及吞吐量。

这里得出的结论可能为如何设计云环境的架构提供一些方向。因此，当前的想法是IaaS更多地利用虚拟机而PaaS开发则利用容器。如果IaaS被设计为使用容器，它们除了可以提供更好的吞吐量和更简单的部署外，也可以减少IaaS和“裸机”系统之间的差异，因为它们支持管理，并提供与本机的几乎相同的性能。但这里的问题是在虚拟机内启动容器，因为这可能与虚拟机的吞吐量开销冲突，与在本地主机上立即启动容器相比，没有任何优点。这些切合实际的想法应该用来考虑对构建和管理IaaS和PaaS系统时技术的选择。多租户是云的另一个非常重要的问题，并且容器或微服务应用程序通常由一些共享相同资源的独立容器中运行的多个服务组成。未来的工作将由共享资源对性能的影响主导。

# 致谢

作者要感谢NeCTAR研究云（www.nectar.org.au）提供的用于执行这些调查的资源。

# 引用

[1] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. Shin, Performance Evaluation of Virtualization Technologies for Server Consolidation, Enterprise Systems and Software Laboratory HP Laboratories Palo Alto HPL-2007-59, 2007.

[2] N. Regola, J. Ducom, Recommendations for virtualization technologies in high performance computing, in: 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CloudCom, 30 2010-dec. 3 2010, pp. 409–416.

[3] N. Slater, Using Containers to Build a Microservices Architecture, viewed 1 April 2015, URL <https://medium.com/aws-activate-startup-blog/usingcontainers-to-build-a-microservices-architecture>.

[4] Docker – Build, Ship, and Run Any App, Anywhere, viewed 1 April 2015, URL <http://www.docker.com>.

[5] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An Updated Performance Comparison of Virtual Machines and Linux Containers, viewed 1 April 2015, URL <http://www.research.ibm.com/>.

[6] D. Merkel, Docker: Lightweight linux containers for consistent development and deployment, Linux J. 2014 (239) (2014).

[7] Flockport, viewed 1 April 2015, URL <http://www.flockport.com>.

[8] P. Rubens, Docker Not the Only Container Option in 2015, IT Business Edge, 2015.

[9] M. Xavier, M. Neves, F. Rossi, T. Ferreto, T. Lange, C. De Rose, Performance evaluation of container-based virtualization for high performance computing environments, in: 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing, (PDP), IEEE, 2013.

[10] S. Soltesz, H. Potzl, M. Fiuczynski, A. Bavier, L. Peterson, Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors, SIGOPS Oper. Syst. Rev. 41 (3) (2007) 275–287.

[11] E.W. Biederman, Multiple instances of the global linux namespaces, in: Proceedings of the Linux, 2006.

[12] P. Menage, Control groups definition, implementation details, examples and api, viewed 1 April 2015, URL http://www.kernel.org/doc/Documentation/ cgroups/cgroups.txt.

[13] L. Marsden, The Microservice Revolution: Containerized Applications, Data and All, viewed 1 April 2015, URL http://www.infoq.com/articles/ microservices-revolution.

[14] A. Polvi, CoreOS is building a container runtime, Rocket, viewed 1 April 2015, URL <http://coreos.com/blog/rocket>.

[15] J. Hwang, S. Zeng, T. Wood, A component-based performance comparison of four hypervisors, in: 2013 IFIP/IEEE International Symposium on Integrated Network Management, (IM 2013), IEEE, 2013.

[16] E. Abdellatief, N. Abdelbaki, Performance evaluation and comparison of the top market virtualization hypervisors, in: 2013 8th International Conference on Computer Engineering & Systems, (ICCES), IEEE, 2013.

[17] S. Varrette, M. Guzek, V. Plugaru, V. Besseron, P. Bouvry, HPC performance and energy-efficiency of Xen, KVM and VMware hypervisors, in: 25th International Symposium on Computer Architecture and High Performance Computing, (SBAC-PAD), IEEE, 2013.

[18] R. Dua, A.R. Raja, D. Kakadia, Virtualization vs containerization to support PaaS, in: 2014 IEEE International Conference on Cloud Engineering, (IC2E), IEEE, 2014.

[19] Z. Estrada, Z. Stephens, C. Pham, Z. Kalbarczyk, R. Iyer, A performance evaluation of sequence alignment software in virtualized environments, in: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGrid), IEEE, 2014.

[20] Heroku, viewed 1 April 2015, URL <https://heroku.com>.

[21] Linux Containers, viewed 1 April 2015, URL <http://linuxcontainers.org>.

[22] PBZIP2, viewed 1 April 2015, URL <http://www.compression.ca/pbzip2/>.

[23] enwik8, viewed 1 April 2015, URL <http://mattmahoney.net/dc/textdata>.

[24] Y-cruncher – A Multi-Threaded Pi-Program, viewed 1 April 2015, URL <http://www.numberworld.org/Y-cruncher>.

[25] Intel⃝R Math Kernel Library – LINPACK Download, viewed 1 April 2015, URL <https://software.intel.com/en-us/articles/intel-math-kernel-librarylinpackdownload>.

[26] LINPACK\_BENCH – The LINPACK Benchmark, viewed 1 April 2015, URL <http://people.sc.fsu.edu/~jburkardt/c_src/linpack_bench/linpack_bench.html>.

[27] Geekbench 3 – Cross-Platform Processor Benchmark, viewed 1 April 2015, URL <http://www.primatelabs.com/geekbench>.

[28] Bonnie++, viewed 1 April 2015, URL <http://www.coker.com.au/bonnie>++.

[29] Sysbench in Launchpad - SysBench: a system performance benchmark, viewed 1 April 2015, URL <https://launchpad.net/sysbench>.

[30] J. McCalpin, STREAM: Sustainable Memory Bandwidth in High Performance Computers, a continually updated technical report (1991-2007), 2015, viewed 1 April URL <http://www.cs.virginia.edu/stream>.

[31] The Netperf Homepage, viewed 1 April 2015, URL <http://www.netperf.org>.

[32] Iperf, viewed 1 April 2015, URL <http://www.iperf.fr>.

[33] Kubernetes, viewed 1 April 2015, URL http://kubernetes.io.