

# Sanal Makineler ve Linux Konteynerlerin Performans Karşılaştırması

**Emre Can Yılmaz<sup>1</sup>, Recai Oktas<sup>2</sup>**

<sup>1</sup> Ondokuz Mayıs Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, Samsun

<sup>2</sup> Ondokuz Mayıs Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Samsun

[ecylmz@bil.omu.edu.tr](mailto:ecylmz@bil.omu.edu.tr), [roktas@bil.omu.edu.tr](mailto:roktas@bil.omu.edu.tr)

**Özet:** Günümüzde, bulut bilişimle birlikte yaygınlaşan tam sanallaştırmadan, hızla popülerliği artan konteyner tabanlı sanallaştırmaya geçişin olduğu gözlemlenmektedir. Bu geçişin nedenleri arasında tam sanallaştırmada kaynakların verimsiz kullanılması ve bunun sonucunda sanal makinelerin iş yükü performansının düşük olması gösterilmektedir.

Bu çalışmada, Linux işletim sistemi çekirdeği özelinde tam sanallaştırma ile konteyner tabanlı sanallaştırmanın başarımlarını kıyaslamalarını yaptık. Yapılan çalışmalar için Linux konteyner yönetim yazılımı olarak Docker ve son zamanlarda Docker'a alternatif olarak ortaya çıkan Rocket kullanırken, tam sanallaştırmada hipervizör olarak KVM (Kernel based Virtual Machine) kullandık. Belirtilen ortamların her biri için imaj boyutu, açılış hızı, işlemci, depolama ve ağ başarımları testleri uygulandı. Bu testlerin sonuçlarının, kullanıcılar için hangi tip sanallaştırmanın hangi koşullarda kendilerine daha uygun olacağı konusunda yardımcı olması beklenmektedir.

**Anahtar Sözcükler:** Bulut Bilişim, Sanal Makine, Linux Konteyner, Hipervizör, Sistem Yönetimi.

## Performance Comparison of Virtual Machines and Linux Containers

**Abstract:** Nowadays, container-based virtualization has become popular. This is why there have been changes in the development process. By developers and system administrators, development and deployment processes are required to minimize their differences. And also differences between developers work environments are required to eliminate. Mentioned these problems, it has a critical role in the development process. In this study, modern solution proposals was made to these problems. Docker is used as container-based virtualization and Ruby on Rails web framework is used as web application.

**Keywords:** Cloud Computing, Virtual Machine, Linux Container, Hypervisor, System Administration.

### 1. Giriş

Bulut bilişimde yaygın olarak sanal makineler kullanılmaktadır. Altyapı olarak servis (IaaS) ve sanal makineler büyük ölçüde eş anlama sahiptir. Amazon EC2 gibi bulut platformların sanal makineleri müşterilerine sunmalarıyla birlikte, müşteriler bu sanal makineler içerisinde veritabanı gibi servislerini çalıştırabilir hale gelmiştir. Pek

çok platform olarak servis (PaaS) ve yazılım olarak servis (SaaS) sağlayıcıları, IaaS üzerine bu sistemleri inşa edip, tüm iş yükünü sanal makinelerin içine yıkmaktadır. Hemen hemen tüm bulut işlemleri sanal makinelerde yapılmaya başlandığından beri, sanal makinelerin performansı önemli bir kriter haline gelmiştir. Bunun sonucu olarak da yerel çalışma zamanı ve sanal makinelerin çalışma zamanları hakkında pek çok çalışma

yapılmıştır [1, 2]. Bu çalışmaların yanı sıra da sanal makinelerin performanslarının artması için de çalışmalar yapılmıştır [3].

Linux içerisinde konteynerleri üretmek için çekirdek isim uzayının gerekli olması yakın zamanda dile gelmiştir [6]. Son üç yıl içerisinde, Linux konteynerleri için, standart bir çalışma zamanı olan, imaj formatı ve inşaa sistemi olarak Docker ortaya çıkmıştır [7]. Docker'a olan ilginin artmasıyla birlikte, kullanılan imaj formatlarının standart hale gelmesi ve imajların güvenliğine dikkat çekmek için Rocket(rkt) ortaya çıkmıştır [8].

Bu çalışmada, popüler olan linux konteynerlerin ve sanal makinelerin, donanım üzerindeki performanslarının yerel ortamdakilere göre kıyası yapılmıştır. Buna ek olarak, hesaplama hızları, bellek bantgenişliği, bellek gecikmesi, ağ bantgenişliği, I/O bant genişliği kıyaslamaları yapılmıştır.

Amacımız birbirinden izole edilmiş, sanal makineleri (özellikle KVM), Linux konteynerleri (Docker ve Rocket) göreceli olarak sanallaştırma yapılmamış Linux işletim sistemiyle kıyaslamaktır. Buna ek olarak sanal makinelerin içerisindeki konteynerlerin performansları incelenmedi. Bunun nedeni sanallaştırma içinde sanallaştırmanın olmasıdır. Linux işletim sistemi, sanal makineleri ve konteynerleri barındırabilmesinden dolayı, elmayı elmayla kıyaslama benzeri bir ortam sunmaktadır.

## 2. Arkaplan

### 2.1 Motivasyon

Unix'in paylaşımlı dosya sistemi, yapılandırma izolasyonunda güvenlik riski oluşturmaktadır. Aynı sunucu üzerinde çalışacak uygulamaların ihtiyaç duyduğu sistem geneli yapılandırmalar çakışabilir. Modern uygulamalar pek çok farklı kitaplığa ihtiyaç duymaktadır. Aynı sunucu üzerinde çalışan uygulamalar aynı kitaplığın farklı

sürümlerine de ihtiyaç duyabilir. Tek işletim sistemi üzerinde birden fazla uygulama çalıştırmak istendiğinde, sistem yönetimi maliyeti yazılım üretim maliyetinden daha fazla olabilmektedir.

Kurumsal firmalarda altyapı ve işyükü aynı firmaya ait olduğundan performans sorunları göz ardı edilebilir durumda olsa da Iaas ve PaaS sağlayıcıları için bu durum böyle değildir. Müşterilerin farklı ihtiyaçları olduğundan ve bir sunucuyu sadece o müşteriye her zaman atayamadıklarından performans kaygıları da artmaktadır. \*aaS sağlayıcıları bu performans sorunu için, CPU, RAM gibi kaynakları birimlere ayırıp, müşterilerin ödedikleri ücretle doğru orantılı kaynak sağlamak fikrini uygulamaktadır. Bahsedilen nedenlerden dolayı, bulut içerisinde sanallaştırma teknolojilerinin kullanılması zorunlu hale gelmektedir.

### 2.2 KVM

Kernel Virtual Machine (KVM) [3] Linux işletim sisteminin bir özelliği olup, Linux süreci içerisinde işletim sistemi çalıştırmayı sağlamaktadır. KVM donanım sanallaştırması teknolojisini kullanarak, karmaşıklığı ve maliyeti düşürmektedir. Buna örnek olarak; ilk hipervizörlerden olan Xen[10] ve VMware [11] ile birlikte, Intel VT-x donanımının kullanılması, karmaşık halka sıkıştırma şemalarını ortadan kaldırdı. KVM, QEMU [12] aracılığıyla I/O cihazlarını ve virtio [13] aracılığıyla da paravirtual I/O cihazlarını desteklemektedir. KVM, canlı göç desteği sağlamakla birlikte, misafir işletim sistemini rahatsız etmeden, fiziksel sunucunun bakımını gerçekleştirebilmeyi sağlar. KVM pek çok yönetim aracı sayesinde kolaylıkla yönetilebilir [15].

KVM üzerinden oluşturulan sanal CPU(vCPU) gerçek CPU sayısından fazla olamaz ve sanal RAM(vRAM)'deki sayfaların her biri, fiziksel RAM'de en fazla bir sayfaya eşleşebilir.

Sanal makineler bir süreçtir ve tüm Linux

kaynak yönetim imkanları sanal makinelere uygulanabilir. Bu hipervizörün yönetimini ve uygulanmasını kolaylaştırırken, misafir işletim sistemi içerisinde kaynak yönetimini zorlaştırmaktadır. Sanal makineler doğası gereği, belirli bir seviyede yalıtım ve kısıtlı arayüzü sayesinde güvenliği sağlamaktadır. Ancak bunlar tüm riskleri ortadan kaldırmaya yetmez. Hipervizörde ortaya çıkabilecek ayrıcalık hakları yönetimindeki açıklıkları, sanal makinelerin misafir işletim sistemine erişmesine neden olabilir.

### 2.3 Linux Konteynerler

Sanal donanımlar üzerinde çalışan işletim sistemlerinin aksine, konteyner tabanlı sanallaştırma varolan işletim sistemini üzerinde çalışır. Genel olarak her sürece konteyner id'si eklenir. Bu konteynerleri sistemdeki diğer kullanıcılar yetkileri dahilinde görüntüleyebilir.

Linux konteynerler, çekirdek isim uzayı özelliğini kullanır. Linux, dosya sistemi, PID, kullanıcı ve ağ isim uzaylarını gerçekler. Örnek verilecek olunursa, her dosya sistemi isim uzayı kendi kök dizinine sahiptir. Tıpkı chroot gibi davranır fakat daha güçlüdür.

İsim uzayları pek çok farklı iş için kullanılabilir fakat genel yaklaşım; dışarıdan görüntülenemeyecek şekilde yalıtılmış konteynerler oluşturmak ve konteyner dışındaki nesnelere erişim sağlanamamasını sağlamaktır. İletim sistemi üzerinde oluşturulan konteynerler aynı çekirdeği paylaşımlarına rağmen, konteynerler içerisinde çalıştırılan süreçler Linux işletim sistemi üzerinde çalışıyormuş gibi görünür. Fakat bu konteynerler birbirinden yalıtıldığından bu durumdan haberdar değildir.

Sanal makinelerin aksine konteynerler sadece bir kaç süreç içerecek şekilde çalıştırılmaktadır. Konteyner içerisinde init, ssh, cron gibi yazılımları birlikte çalıştırıldığında sistem konteyneri şeklinde

adlandırılırken, sadece uygulama çalıştığında uygulama konteyneri şeklinde adlandırılmaktadır.

Cgroups Linux çekirdeğinin bir grup sürecin kullanacağı kaynak miktarını sınırlama yöntemidir [16]. Genellikle, konteynerlerin kullanabileceği CPU ve RAM miktarını ayarlamakta kullanılır. Yine cgroups sayesinde basit bir şekilde sistemdeki mevcut konteynerlere kaynak eklenebilir. Bunun yanı sıra konteyner içerisindeki bir süreci sonlandırmak için cgroups güvenilir bir yol sunmaktadır. Çünkü, Linux işletim sadece bir çekirdeğe sahiptir ve bu çekirdek tüm konteynerlerin içeriğini görebilme yeteneğine sahiptir.

Konteynerlerin kaynak yönetiminde hala var olan sorunlardan biri, konteyner kendisine ne kadar kaynak sınırı koyulduğunu bilmiyor, tüm sistem kaynaklarını kendisinin sanıyor. Örnek vererek söylenirse; konteynere sadece 1 CPU atanmış olmanıza rağmen konteyner çalıştığı sistem üzerindeki tüm CPU'ları kendisinin sanmaktadır. Aynı durum RAM için de geçerlidir. Eğer uygulama kendini sistemdeki mevcut RAM'e göre düzenliyorsa bu uygulamada tutarsızlıklara/hatalara neden olabilir.

Linux konteynerleri için pek çok yönetim aracı mevcuttur. Bunlardan başlıca olanları, LXC[18], systemd-nspawn [19], Docker ve Rocket'tir. Pek çok özelliklerinden ve kolay kullanımından dolayı Docker popülerliğini hızla artırmaktadır. Docker ile yaygınlaşan konteyner çözümlerine ek olarak Rocket ortaya çıkmıştır. Docker'dan farklı olarak güvenliğe önem vermesi ve uygulama konteynerlerini standart haline getirmek hedeflenmiştir. Bunun yanı sıra Docker'dan farklı olarak systemd-nspawn konteyner yönetim kütüphanesini kullanmaktadır. Dosya sistemindeki katmanlı yapısından dolayı, imaj üretimindeki katmanları tekrar kullanması sanal makine imajlarına göre diskte daha az yer kaplamasına neden olmaktadır. Bu da uygulama konuşlandırma sürecini

hızlandırmaktadır.

Bu çalışmada başlangıç süreleri kıyası için bir ölçüm yapılmamıştır. Bu alanda yapılan çalışmalar şüpheye ihtimal vermeyecek şekilde konteynerlerin daha hızlı başladığını göstermektedir [20].

### 3. Değerlendirme

Performans değerlendirmede pek çok yaklaşım vardır. Bu çalışmada, sanallaştırma teknolojileri ile fiziksel makinedeki(sanallaştırma olmayan) maksimum işyükü altındaki durumlarını inceledik. Bunun sonucunda da aynı donanımına sahip birden fazla fiziksel makineye ihtiyaç duyuldu.

Yapılan testlerin tamamı; Lenovo C560 57324249 modeli, Intel® Core™ i5-4570T CPU @ 2.90GHz × 4 işlemcili toplamda 4 çekirdeği bulunan, toplamda 12 GB RAM'e sahip sunucu üzerinde gerçekleştirildi. Bu sunucu üzerinde Ubuntu 14.04(Trusty) 64 bit, 3.13.0-62 Linux çekirdeği, Docker 1.8.2, QEMU 2.0.0, libvirt 1.2.2, rkt 0.5.5 kullanılmıştır.

Docker ve rkt konteynerlerine cgroups aracılığıyla herhangi bir sınırlama konulmamıştır. CPU, bellek, ağ ve depolama konularında mikro kıyaslama betikleri kullandık.

### 3.1 CPU

Sıkıştırma bulutta sıklıkla kullanılan bileşenler için uygulanan bir yöntemdir.PXZ [21], LZMA algoritmasını kullanan, paralel ve kayıpsız veri sıkıştırma aracıdır. Bu çalışmada PXZ programının 4.999.9beta sürümü kullanılmıştır. 2. seviye sıkıştırma tekniği kullanıldı.

Tablo 2'de gözüktüğü gibi, Docker, rkt ve Linux'ta sonuçlar çok yakınken, ayarsız KVM %23 daha yavaştır.

### 3.2 HPC

“High Performance Computing” yani yüksek başarımlı hesaplama testidir. Linpack yoğun lineer denklem sistemlerini LU faktörizasyonu algoritmasını kullanarak çözer [22]. Hesaplama işlemlerinin de zamanın büyük çoğunluğu, kayan nokta sayılarının bir vektörle çarpılıp diğer bir vektöre eklenmesine harcanır. Çalışmada, Intel Math Kernel Library (MKL) tabanlı Linpack programını kullandık [23].

Tablo 2, Linux, Docker, rkt, KVM üzerindeki Linpack'in performansını göstermektedir. Sonuçlarda Linux, Docker ve rkt'nin performanslarının neredeyse aynı olduğu gözlemlenmektedir. Ancak ayarlama yapılmamış KVM'deki durumun diğerlerine göre kötü olduğu gözükmemektedir. eğer KVM'e fiziksel makinenin sahip olduğu CPU'lar ve topolojisi gösterilirse sonuçların nispeten daha iyi olduğu görülmektedir. Ortaya çıkan sonuçlar beklenen sonuçlardır.

### 3.3 Bellek Bant Genişliği

Bellek bantı genişliği için “Stream” testi kullanılmıştır. Stream testi vektörler üzerinde basit işlemler yaparak bellek bant genişliği ölçümü yapmaktadır [22]. Bant genişliği, bir iletim ortamının ya da haberleşme kanalının kapasitesini ifade etmek için kullanılır.

| İsim  | Kernel                   | Bytes per iteration | FLOPS per iteration |
|-------|--------------------------|---------------------|---------------------|
| COPY  | $a[i] = b[i]$            | 16                  | 0                   |
| SCALE | $a[i] = q * b[i]$        | 16                  | 1                   |
| ADD   | $a[i] = b[i] + c[i]$     | 24                  | 1                   |
| TRIAD | $a[i] = b[i] + q * c[i]$ | 24                  | 2                   |

**Tablo 1.** Stream bileşenleri

Başka bir deyişle bir kanal üzerinde taşınabilecek en fazla frekansa sahip sinyal, kanalın bant genişliğidir. Bant genişliği ne kadar büyükse, belli bir süre içinde aktarılacak verinin hacmi de o kadar büyük olur [24].

| Uygulanan Test      |       | Linux   | Docker  | rkt     | KVM     |
|---------------------|-------|---------|---------|---------|---------|
| PXZ (MB/s)          |       | 7,94    | 7,00    | 7,35    | 6,2     |
| Linpack (GFLOPS)    |       | 26,94   | 26,86   | 25,93   | 18,49   |
| RandomAccess (GUPS) |       | 0,0550  | 0,0515  | 0,0520  | 0,0545  |
| Stream (GB/s)       | Add   | 11.8211 | 11.8406 | 11.7503 | 11.5224 |
|                     | Copy  | 11.4309 | 11.4464 | 11.1776 | 11.2618 |
|                     | Scale | 11.3928 | 11.4289 | 11.1505 | 11.2498 |
|                     | Triad | 11.7716 | 11.7792 | 11.7169 | 11.513  |

**Tablo 2.** PXZ, Linpack, Stream, RandomAccess testleri sonuçlarının aritmetik ortalamaları

Aslında performansı, gecikme değil bellek bant genişliği etkiler. Test dört bileşenden oluşmaktadır ve bunlar tablo 2’de açıklanmıştır.

Tablo 2, Linux, Docker, rkt ve KVM ortamlarında Stream performansını göstermektedir. Tüm bu ortamlarda neredeyse yakın sonuçlar elde edilmiştir.

### 3.4 Rastgele Bellek Erişimi

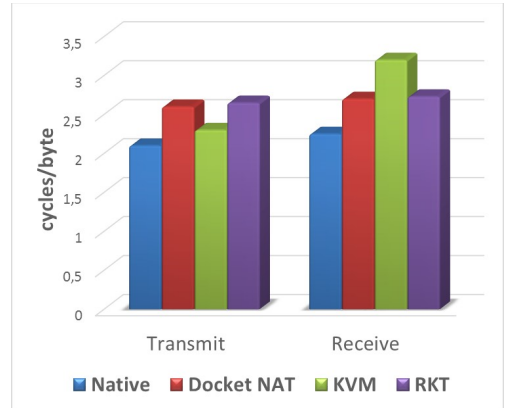
Stream testinde düzenli bir şekilde bellekten veri çekerek bu test uygulansa da Random Access testinde [22] veriler bellekten rastgele seçilerek test gerçekleştirilir. Test, çalışma kümesi olarak önbelleğin veya TLB’nin erişemeyeceği kadar belleğin büyük bir kısmını seçerek başlar. Bellekten 8 bitlik okuma düzenleme ve yeniden yazma işlemleri yapılır. Belleğe erişilirken kullanılan rastgele konumlar, bellek harcaması gerektirmeyen “linear feedback shift register” yöntemiyle belirlenmiştir.

Stream testindeki gibi TLB’nin erişemeyeceği kadar büyük veriyle çalışılmıştır. Bunun nedeni, rastgele bellek erişimleri TLB’nin erişebileceğinden daha büyük paternlerde çalışır.

Tablo 2’de de görüldüğü gibi tüm platformlarda sonuçlar çok yakın çıkmıştır. Birbirlerinden bariz bir üstünlükleri bulunmamaktadır.

### 3.5 Ağ Bant Genişliği

Ağ bant genişliği testi için nuttcp [25] aracını kullandık. Şekil 1’de görüldüğü gibi, Docker ve rkt konteynerleri ağa NAT kullanarak bağlar. KVM yapılandırmasında virtio kullanılmıştır. Tüm platformlarda ulaşılan maksimum hız eşit çıkmıştır.

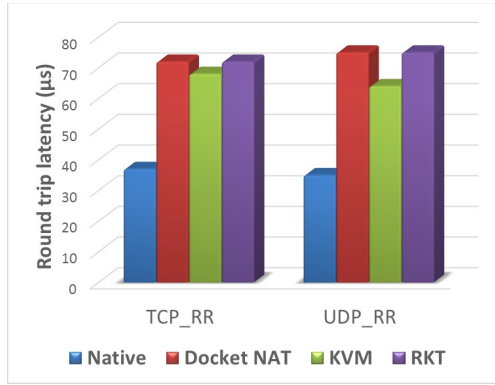


**Şekil 1.** TCP toplu transfer verimi (CPU cycles / byte)

Şekil 1’de bu ölçümler sırasında CPU kullanım verimliliği sonuçları gözlemlenmiştir. Bu ölçümlerde “perf stat -a” kullanılmıştır. KVM veri iletimi işlemi sırasında daha verimli olurken, Docker’da önce köprü sonra NAT işlemi yapıldığından KVM’e göre verimi düşüktür. Ancak KVM veri alımı işlemi sırasında diğerlerine göre çok daha fazla işlem yükü oluşturmaktadır.

### 3.6 Ağ Gecikmesi

Ağ gecikmesi testi için netperf istek-yanıt aracını kullandık. Bu test sırasında sunucularda birinde netperf sunucusu(netserver) çalışırken, diğer sunucuda ise netperf istemcisi çalışır. İstemci 100 baytlık istekler gönderirken, sunucu 200 baytlık yanıtlar gönderir. İstemci diğer isteği göndermeden önce sunucudan gelecek yanıtı bekler. Belirtilen bu adımlar 1 işlem olarak kabul edilir.

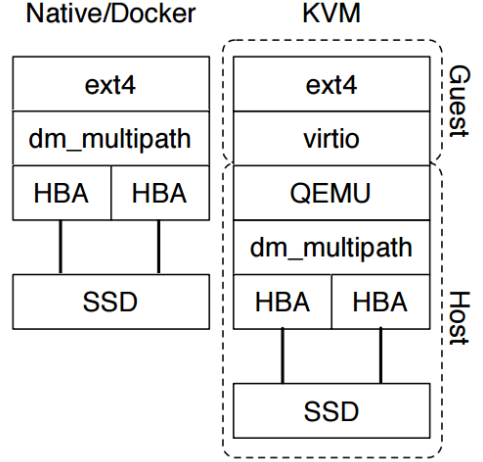


Şekil 2. Ağ round-trip gecikmesi (µs).

Şekil 2’de yapılan işlemlerin ölçüm sonuçları görülmektedir. Docker’da ve rkt’de kullanılması gecikmeyi yaklaşık 2 katına çıkarmıştır. KVM de ise sanallaştırma olmayan sunucuya göre yaklaşık %80 oranında daha fazladır.

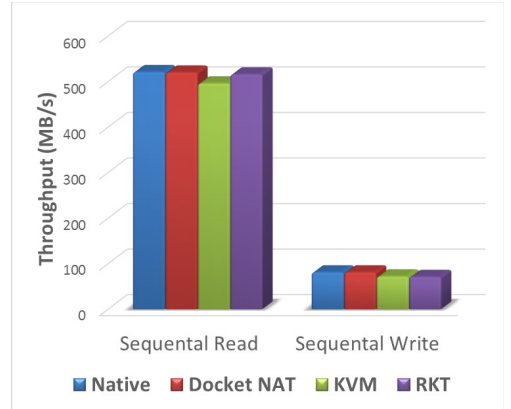
### 3.7 Blok I/O İşlemleri

Blok I/O işlemlerinde kullanmak üzere 120 GB SSD Samsung EVO 850 model disk kullandık. Dosya sistemi olarak öntanımlı ayarlarıyla ext4 dosya sistemi kullandık. Sanal makine sisteme bağlanan disk kullanmak için “virtio”yu kullanmaktadır. Bu yapılandırma şekli şekil 3’te gösterilmiştir. Testi gerçekleştirmek için fio(v2.1.3) [26] aracı kullanıldı.



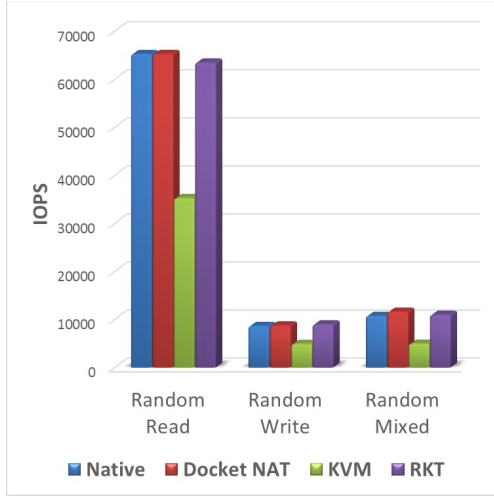
Şekil 3. fio için kullanılan depolama konfigürasyonları

Şekil 4’de sıralı okuma yazma performansları gösterilmiştir. Performansların hemen hemen aynı olduğu gözlemlenmektedir.



Şekil 4. Sıralı I / O verimliliği (MB/s).

Şekil 6’da ise 4kB’lık bloklar halinde rastgele okuma yazma performanslarının sonuçları gösterilmiştir. Beklenildiği gibi Docker’da ve rkt’de herhangi bir performans sorunu yokken, KVM her bir i/o işlemi için QEMU’ya başvurduğundan performans kaybı yaşamaktadır.



Şekil 5. Rastgele I / O verimliliği (IOPS).

### 3. Kaynak Kodlar

Bu çalışmada çalıştırılan testler <https://github.com/ecylmz/docker-rkt-kvm-comp/> adresinde bulunmaktadır.

### 4. Sonuç ve Öneriler

Donanım ve yazılımların son on yılda gelişmesiyle birlikte, Linux konteynerler ve sanal makineler olgunlaşmış teknoloji haline gelmişlerdir. Docker ve rkt her açıdan sanal makinelerin performansına erişebildiği gözlemlenmiştir.

Çalışmalarımız neticesinde KVM'in yoğun I/O işlemlerinde ağır kaldığı gözlemlenmiştir. Yüksek I/O işlemlerinin yapılacağı uygulamalar için çok da uygun olmadığı anlaşılmıştır.

Sanal makine içerisinde konuşlandırılan konteynerlerin, sanallaştırma yapılmamış Linux ile kıyaslamasının bir faydası olmadığı açıklanmıştır. Eğer sanal makineler üzerinde konteynerler çalıştırılmak istenirse, bu ekstra güvenlik katmanı yaratmasına karşın, QEMU'ya yetkisiz erişim sağlayan biri aynı zamanda konteyner içerisine de erişim sağlamaktadır.

Docker'a alternatif olarak çıkan rkt'nin beklenildiği gibi eşit performans sergilediği gözlemlenmiştir.

Bu çalışmada, tek bir sunucu üzerine sadece bir sanal makine veya sadece bir konteyner konulmuştur. Gerçek hayatta sunucular genellikle bu şekilde kullanılmaz. Bulutta genellikle sağlayıcılar bu sunucuları daha küçük birimlere kullanarak müşterilerine sunar. Ayrıca bulut bilişimde kullanılan sunucuların bu çalışmada kullanılan sunuculardan çok daha üstün performansa sahip olması gerekmektedir. Bu çalışmada orataya çıkan sayısal değerler, platformların birbirlerine göre oransal olarak kıyası söz konusudur.

### 5. Kaynaklar

- [1] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, and James Owens. Quantifying the performance isolation properties of virtualization systems. In Proceedings of the 2007 Workshop on Experimental Computer Science, ExpCS '07, 2007.
- [2] Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G Shin, et al. Performance evaluation of virtualization technologies for server consolidation. HP Labs Technical Report, 2007.
- [3] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: the Linux virtual machine monitor. In Proceedings of the Linux Symposium, volume 1, pages 225–230, Ottawa, Ontario, Canada, June 2007.
- [4] Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavier, and " Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In Proceedings of the 2nd ACM SIGOPS/EuroSys European

Conference on Computer Systems 2007, EuroSys '07, pages 275–287, 2007.

[5] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom. The Use of Name Spaces in Plan 9. In Proceedings of the 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring, pages 1–5, 1992.

[6] E. W. Biederman. Multiple instances of the global Linux namespaces. In Proceedings of the 2006 Ottawa Linux Symposium, 2006.

[7] Solomon Hykes and others. What is Docker? <https://www.docker.com/whatisdocker/>. (Ziyaret Tarihi: 5 Ağustos 2015)

[8] Rocket – <https://coreos.com/blog/rocket/> (Ziyaret Tarihi: 5 Ağustos 2015)

[10] Xen Project Software Overview. [http://wiki.xen.org/wiki/Xen\\_Overview](http://wiki.xen.org/wiki/Xen_Overview).

[11] Virtualization Overview. <http://www.vmware.com/pdf/virtualization.pdf>. (Ziyaret Tarihi: 1 Ağustos 2015)

[12] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[13] Rusty Russell. Virtio: Towards a de-facto standard for virtual I/O devices. SIGOPS Oper. Syst. Rev., 42(5):95–103, July 2008.

[15] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehorster, " and Andre Brinkmann. Non-intrusive virtualization management using ' libvirt. In Proceedings of the Conference on Design, Automation and Test in Europe, DATE '10, pages 574–579. European Design and Automation Association, 2010.

[16] KAP Temelli Özgür Sanallaştırma Çözümleri [https://seminer.linux.org.tr/wp-content/uploads/kap\\_temelli\\_ozgur\\_sanallastirma\\_cozumleri.pdf](https://seminer.linux.org.tr/wp-content/uploads/kap_temelli_ozgur_sanallastirma_cozumleri.pdf)] (Ziyaret Tarihi: 10 Ağustos 2015)

[18] Stphane Graber and others. LXC—Linux containers. <https://linuxcontainers.org/>.

[19] Lennart Poettering, Kay Sievers, and Thorsten Leemhuis. Control centre: The systemd Linux init system. <http://www.h-online.com/open/features/Control-Centre-The-systemd-Linux-init-system-1565543.html>, May 2012.

[20] Boden Russell. KVM and Docker LXC Benchmarking with OpenStack. <http://bodenr.blogspot.com/2014/05/kvm-and-docker-lxc-benchmarking-with.html>, May 2014.

[21] PXZ—parallel LZMA compressor using liblzma. <https://jnovy.fedorapeople.org/pxz/>, 2012.

[22] J. Dongarra and P. Luszczyk. Introduction to the HPCChallenge Benchmark Suite. Technical report, ICL Technical Report, 10 2005. ICL-UT-05-01.

[23] Intel Math Kernel Library—LINPACK Download. <https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download>.

[24] Bant Genişliği [https://tr.wikipedia.org/wiki/Bant\\_geni%C5%9Fl%C4%9Fi](https://tr.wikipedia.org/wiki/Bant_geni%C5%9Fl%C4%9Fi) (Ziyaret Tarihi: 10 Ağustos 2015)

[25] The “nuttcp” Network Performance Measurement Tool. <http://www.nuttcp.net/>.

[26] Jens Axboe. Flexible IO Tester. <http://git.kernel.dk/?p=fio.git;a=summary>.