مقاله پژوهشی

بررسی و ارزیابی عملکرد وب‌سرورهای Apache و Nginx بر بستر کانتینرهای داکر، پادمن و LXC

علی فرهادیان، مصطفی بستام، احسان عطایی و مهدی باباگلی

چکیده: گسترش خدمات ابری، ضرورت بهره‌گیری از روش‌های مجازی‌سازی به‌منظور استفاده بهینه از منابع سخت‌افزاری را افزایش داده است. در گذشته، ماشین‌های مجازی گزینه اصلی برای مجازی‌سازی بودند، اما با ظهور کانتینرها، امکان حذف سیستم‌عامل اضافه و کاهش سربار منابع فراهم شد. فناوری‌هایی مانند داکر، پادمن و LXC در این حوزه کاربرد گسترده‌ای پیدا کرده‌اند. در همین راستا، وب‌سرورهای Nginx و Apache نیز برای سازگاری با این فناوری‌ها بهینه‌سازی شده‌اند. در این مقاله، عملکرد این دو وب‌سرور بر بستر کانتینرهای مختلف و تحت شرایط گوناگون منابع و همروندی بررسی شده است. آزمایش‌ها نشان می‌دهند که انتخاب نوع کانتینر به نوع وب‌سرور و میزان منابع بستگی دارد. در محیط‌های با منابع محدود، استفاده از LXC برای Apache نتایج بهتری داشته است. در مقابل، در شرایط با منابع بیشتر، داکر برای اجرای Nginx عملکرد مطلوب‌تری ارائه کرده است. یافته‌های این پژوهش می‌تواند به تصمیم‌گیری بهتر در انتخاب ترکیب مناسب کانتینر و وب‌سرور بر اساس نیازهای زیرساختی کمک کند.

کلیدواژه: **مجازی‌سازی، کانتینرها، داکر، Apache، Nginx، LXC.**

۱- مقدمه

امروزه با توجه به رشد خدمات مبتنی بر ابر از جمله محاسبات ابری[۱]، ذخیره‌سازی ابری و برنامه‌های تحت ابر، اهمیت شناخت و استفاده از روش‌های مجازی‌سازی بیش از پیش احساس می‌شود. مقصود از مجازی‌سازی در روش‌های قدیمی‌تر، استفاده از یک هایپروایزر بین زیرساخت ماشین فیزیکی میزبان و سیستم‌عامل‌های اجراشده بر روی آن بوده است. در این روش، یک سیستم‌عامل (کرنل) همراه با برنامه و کتابخانه‌های وابسته به آن به عنوان یک ماشین مجازی[۲] بر روی یک لایه هایپروایزر[۳] پیاده‌سازی می‌شود. این روش سنتی به علت وجود یک لایه سیستم‌عاملی دیگر در ماشین مجازی، منابع بیشتری را استفاده می‌نماید [۱]. این سربار، بیشتر در مواقعی همچون مهاجرت در محیط‌های ابری بسیار قابل توجه خواهد بود، زیرا لازم است برای مهاجرت یک ماشین مجازی تمامی لایه‌های آن ماشین مجازی اعم از کرنل سیستم‌عامل به علاوه وابستگی‌ها و خود برنامه به عنوان یک واحد جابه‌جا شود. این روش مجازی‌سازی به علت استفاده از کرنل مجزا و عدم استفاده از کرنل مشترک، از امنیت بالاتری بهره می‌برد. این نوع پیاده‌سازی، نقش کاربردی و ویژه‌ای در معماری‌های ابر ایفا می‌کند [۲]. روش مجازی‌سازی مبتنی بر ماشین‌مجازی، با توجه به روش پیاده‌سازی به دو نوع کلی تقسیم می‌شود. در روش اول که به آن هایپروایزر نوع یک[۴] نیز اطلاق می‌شود، هایپروایزر مستقیم بر روی سخت‌افزار نصب می‌شود و به مدیریت ماشین‌های مجازی می‌پردازد. در روش دوم با عنوان هایپروایزر نوع دو، هایپروایزر بر روی سیستم‌عامل نصب‌شده بر روی سخت‌افزار نصب می‌شود. VMware ESX، Xen و Hyper-v از جمله روش‌های مجازی‌سازی هایپروایزر نوع یک و Vmware Workstation و KVM[۵] از جمله روش‌های مجازی‌سازی هایپروایزر نوع دو محسوب می‌شوند [۳]. شایان ذکر است که KVM با تبدیل هسته لینوکس به یک هایپروایزر، ساختاری متمایز از سایر معماری‌های نوع دوم ارائه می‌دهد. [۴]. با توجه به سربار زیاد روش‌های سنتی مجازی‌سازی، امروزه روش‌های مجازی‌سازی جدیدی تحت عنوان مجازی‌سازی مبتنی بر کانتینر[۶] ارائه شده‌اند [۵]. در این روش با حذف لایه سیستم‌عامل در معماری‌های مجازی‌سازی، هر کانتینر به صورت مستقیم، کرنل سیستم‌عامل میزبان را به صورت اشتراکی استفاده می‌کند. در این ساختار، هر واحد به یک برنامه به همراه کتابخانه‌های مورد نیاز آن اشاره دارد. این روش با حذف سیستم‌عامل مجزا برای هر ماشین مجازی و معرفی آن به عنوان کانتینر، سربار و حجم هر واحد مجزا را کاهش می‌دهد. این واحدها از قابلیت راه‌اندازی، توقف و مهاجرت با سرعت بالاتری برخوردارند که حاکی از بهبود قابل توجه عملکرد آنها در مقایسه با روش‌های سنتی است. البته این روش چالش‌هایی را نیز به‌دنبال دارد. به دلیل استفاده از کرنل اشتراکی در این روش، ایزوله‌سازی ضعیف‌تری در مقایسه با روش‌های مبتنی بر ماشین مجازی وجود دارد. مدیریت میزان و نحوه دسترسی هر پردازه به منابع سخت‌افزاری و نرم‌افزاری، ایزوله‌سازی نام دارد که موجب جداسازی کلیه منابع یک پردازه از دیگر پردازه‌ها می‌شود [۶]. شکل ۱ شماتیک کلی مربوط به کانتینرسازی و روش‌های سنتی مجازی را نشان می‌دهد.

این مقاله در تاریخ ۱۱ دی ماه ۱۴۰۳ دریافت و در تاریخ ۴ خرداد ماه ۱۴۰۴ بازنگری شد.

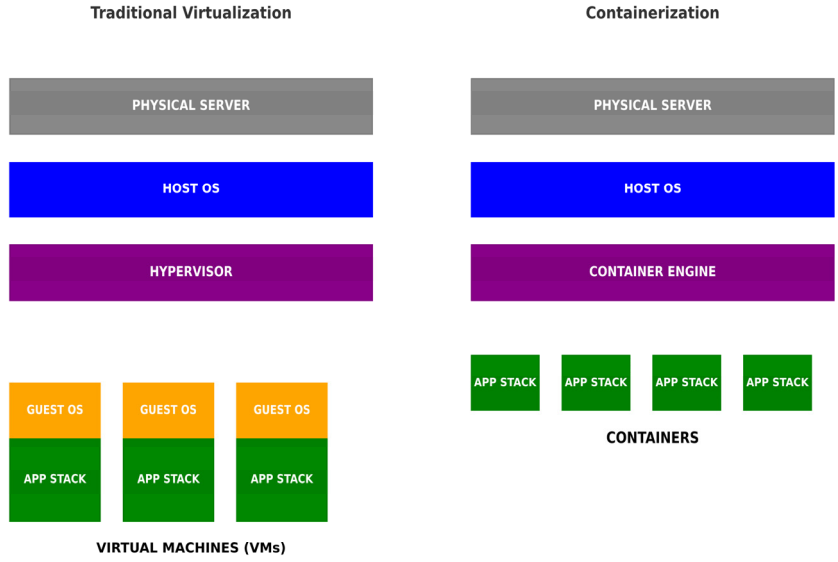
علی فرهادیان، دانشکده مهندسی کامپیوتر، دانشگاه مازندران، بابلسر، مازندران، (email: alifrd49@gmail.com).

مصطفی بستام (نویسنده مسئول)، دانشکده مهندسی کامپیوتر، دانشگاه مازندران، بابلسر، مازندران، (email: bastam@umz.ac.ir).

احسان عطایی، دانشکده مهندسی کامپیوتر، دانشگاه مازندران، بابلسر، مازندران، (email: ataei@umz.ac.ir).

مهدی باباگلی، دانشکده مهندسی کامپیوتر، دانشگاه مازندران، بابلسر، مازندران، (email: ataabagoli@email.kntu.ac.ir).

1. Cloud Computing
2. Virtual Machine
3. Hypervisor
4. Type-1 Hypervisor
5. Kernel-Based Virtual Machine
6. Container



شکل ۱: شماتیک کلی کانتینرسازی و مجازی‌سازی سنتی.

روش‌های کانتینری به علت سربار کمتر و امکان به اشتراک‌گذاری کتابخانه‌ها مابین کانتینرها، به سرعت در محیط‌های ابری محبوب و رایج شده‌اند. کانتینر، زمینه‌ساز ظهور معماری جدیدی با نام مایکروسرویس شده است [۷]. به مرور زمان با هدف بهبود کارایی و کم‌کردن سربار، بالابردن سرعت و بهبود امنیت، فناوری‌های کانتینری متفاوتی از جمله پادمن[۱]، داکر[۲] و LXC[۳] ارائه شده‌اند. علاقه جوامع متن‌باز[۴] به کانتینر، به‌ویژه داکر و نیز همکاری جهت بهبود و یکپارچه‌سازی آن با محیط ابری باعث شد تا داکر در مقایسه با سایر روش‌های کانتینرسازی از محبوبیت و استفاده بیشتری برخوردار شود. بخش مهمی از این پیشرفت، به علت قرارگرفتن داکر در معماری‌های تحت ابر و ابزارهای Orchestration (مدیریت‌کردن خودکار اجرا و هماهنگ‌سازی کانتینرها) مانند KAS[۵] و Docker Swarm بوده است. قابلیت مدیریت گسترده کانتینرها از طریق سیستم‌های جامع، جایگاه داکر را در معماری‌های ابری تقویت کرده است [۸]. از طرف دیگر، وب‌سرورها به عنوان یکی از مهم‌ترین استفاده‌کنندگان معماری مایکروسرویس، مستعد بهره‌مندی از مجازی‌سازی مبتنی بر کانتینر به شمار می‌روند [۹]. در این مقاله، عملکرد دو مورد از معروف‌ترین و پرکاربردترین وب‌سروهای موجود یعنی Nginx و Apache به منظور یافتن بهترین روش کانتینرسازی در بستر داکر، پادمن و LXC مورد بررسی قرار خواهد گرفت. برای رسیدن به این مهم، ابتدا معماری روش‌های کانتینرسازی بررسی می‌شود. در گام بعد، عملکرد و ساختار وب‌سرورها تحلیل خواهد شد و سپس با توجه به اطلاعات به‌دست‌آمده، عملکرد استفاده از فناوری‌های کانتینرسازی در وب‌سرورها مورد واکاوی قرار خواهد گرفت.

در ادامه نحوه سازماندهی این مقاله شرح داده شده است. بخش دوم به مرور فناوری‌های کانتینری و معماری وب‌سرورهای Apache و Nginx اختصاص دارد تا زمینه مفهومی و فنی پژوهش مشخص شود. بخش سوم، معماری طراحی‌شده برای کانتینرها، پیکربندی منابع و سناریوهای هم‌روندی مورد استفاده را تشریح می‌کند. بخش چهارم نیز به تحلیل نتایج حاصل از پیاده‌سازی و ارزیابی عملکرد وب‌سرورها در شرایط مختلف منابع و همزمانی می‌پردازد. در ادامه، بحث کلی درباره مزایا و معایب ترکیب‌های مختلف کانتینر و وب‌سرور ارائه می‌شود. بخش پایانی نیز به نتیجه‌گیری و ارائه پیشنهادهایی برای انتخاب ترکیب مناسب کانتینر و وب‌سرور با توجه به نیازهای زیرساختی اختصاص دارد. به طور کلی اهداف این پژوهش به شرح زیر می‌باشد:

- تحلیل و مقایسه عملکرد وب‌سرورها در شرایط منابع محدود و همروندی بالا: انجام بررسی جامع نحوه عملکرد وب‌سرورهای Apache و Nginx در بسترهای مختلف کانتینری (داکر، پادمن و LXC) تحت شرایط گوناگون منابع سخت‌افزاری و سطوح مختلف همزمانی
- ارزیابی تأثیر معماری وب‌سرورها بر عملکرد آنها در کانتینرها: مقایسه معماری‌های متفاوت پردازش، شامل مدل پردازه/ نخ[۶] در Apache و مدل مبتنی بر رویداد[۷] در Nginx و تحلیل نقاط قوت و و ضعف هر کدام در محیط کانتینری
- ارائه راهکارهای بهینه برای استفاده از کانتینرها در شرایط مختلف: پیشنهاد ترکیب‌های بهینه وب‌سرور و فناوری کانتینری متناسب با شرایط زیرساختی مختلف؛ به‌عنوان نمونه، استفاده از LXC در محیط‌های با منابع محدود و داکر در محیط‌های دارای منابع بیشتر
- بررسی مزایا و محدودیت‌های فناوری‌های مختلف کانتینرسازی: تحلیل ویژگی‌ها و نقاط قوت و ضعف هر یک از فناوری‌های داکر، پادمن و LXC به‌منظور کمک به انتخاب آگاهانه و متناسب با نیاز کاربران در طراحی زیرساخت وب‌سرور

۲- ادبیات موضوعی و تعاریف

برای تحلیل عملکرد کانتینرها در بستر وب‌سرورها، آشنایی با برخی مفاهیم کلیدی در سیستم‌عامل‌های لینوکس و یونیکس ضروری می‌باشد. دو مفهوم مهم در این زمینه، فضای نام‌ها[۸] و گروه‌های کنترل[۹] هستند که که نقش اساسی در ایزوله‌سازی منابع ایفا می‌کنند. فضای نام‌ها به گونه‌ای

1. Podman
2. Docker
3. Linux Containers
4. Open Source
5. Kubernetes
6. Process/Thread
7. Event-Loop
8. Namespace
9. Cgroups

```
root@ubuntu-nginx:~# ps -eaf | grep nginx
root         279       1  0 12:32 ?        00:00:00 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
www-data     280     279  0 12:32 ?        00:04:53 nginx: worker process
www-data     281     279  0 12:32 ?        00:05:05 nginx: worker process
root        2398     734  0 14:33 pts/1    00:00:00 grep --color=auto nginx
root@ubuntu-nginx:~# cat /proc/280/limits
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             unlimited            unlimited            processes
Max open files            1024                 1048576              files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       63154                63154                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
root@ubuntu-nginx:~#
```

شکل ۲: اطلاعات مربوط به محدودیت‌های مربوط به یک پردازه.

طراحی شده‌اند که دیدگاه و دسترسی هر پردازش را نسبت به سایر پردازش‌ها و منابع سیستم محدود می‌کنند. به‌عنوان مثال فضای نام PID برای جداسازی فرایندها، فضای نام Network برای تعریف شبکه‌های مجازی و فضای نام Mount برای محدودکردن دسترسی به فایل سیستم به کار می‌رود. در کنار آن، Cgroup با مدیریت منابعی مانند CPU و حافظه، امکان تخصیص دقیق منابع به هر پردازش را فراهم می‌سازد. ترکیب این دو فناوری، پایه‌ای برای ساخت کانتینرها فراهم کرده و اجرای ایزوله‌شده برنامه‌ها را ممکن می‌سازد. این ایزوله‌سازی به طور مستقیم باعث بهبود امنیت و کارایی در محیط‌های کانتینری می‌شود [۱۰]. در مجموع، این مفاهیم به کانتینرها اجازه می‌دهند تا با سربار کمتر و کارایی بیشتر نسبت به ماشین‌های مجازی عمل کرده و زیرساخت مناسبی برای معماری‌های مدرن مبتنی بر کانتینر فراهم آورند.

۲-۱ ایزوله‌سازی پردازه‌ها

ایزوله‌سازی پردازه‌ها با فضای نام‌ها و ابزارهای کرنل لینوکس در سیستم‌های کانتینری انجام می‌شود. این فرایند، منابع هر پردازه را از سایر پردازه‌ها جدا کرده و مانع از دسترسی یا تداخل آنها می‌شود؛ موضوعی که نقش مهمی در افزایش امنیت و بهینه‌سازی عملکرد سیستم دارد [۶]. یکی از ابزارهای مهم در این زمینه Sysctl است که به تنظیم متغیرهای کرنل می‌پردازد و امکان شخصی‌سازی سیستم‌عامل را فراهم می‌آورد. ابزار دیگر، Ulimit، برای محدودسازی منابع هر پردازه مانند تعداد فایل‌های باز یا حجم حافظه در دسترس کاربرد دارد. با استفاده از این تنظیمات، سیستم‌عامل قادر به مدیریت میزان استفاده از منابع پردازه‌ها و جلوگیری از بروز تداخل بین آنها است [۱۱]. در سیستم‌های کانتینری، ایزوله‌سازی پردازه‌ها به دو روش اصلی انجام می‌شود: ۱) با استفاده از قابلیت‌های بومی سیستم‌عامل لینوکس مانند Cgroup و فضای نام‌ها که منابع را به‌صورت مستقیم محدود کرده و محیطی ایزوله برای هر پردازه ایجاد می‌کنند. ۲) از طریق دایمن‌ها[1] که درخواست‌های سیستمی پردازه‌ها پردازه‌ها را پیش از رسیدن به کرنل مدیریت می‌کنند. مثلاً LXC با بهره‌گیری از ابزارهای داخلی لینوکس این محدودیت‌ها را به‌صورت مستقیم اعمال می‌کند؛ در حالی که داکر از یک دایمن مرکزی برای کنترل منابع و مدیریت ایزوله‌سازی استفاده می‌کند. در هر دو روش، پردازه‌ها در محیطی ایزوله و مدیریت‌شده اجرا می‌شوند، به گونه‌ای که عملکرد آنها مستقل از سایر پردازه‌ها خواهد بود. دایمن به برنامه‌ای در سیستم‌عامل‌های مبتنی بر یونیکس و لینوکس می‌گویند که در پس‌زمینه اجرا می‌شود و به‌طور خودکار به وظایف خاص پاسخ می‌دهد، بدون آنکه نیاز به تعامل مستقیم با کاربر داشته باشد. این برنامه‌ها معمولاً مسئول ارائه خدمات سیستمی مانند مدیریت شبکه، منابع یا اجرای زمان‌بندی‌شده وظایف هستند.

ابزار ulimit در سیستم‌عامل‌های یونیکس و لینوکس برای کنترل و محدودسازی منابع قابل استفاده توسط پردازه‌ها به‌کار می‌رود. این ابزار با استفاده از دو پارامتر soft limit و hard limit، سقف مصرف منابع را برای هر پردازه را تعیین می‌کند. مقدار soft limit حداکثر میزان منابعی است که پردازه می‌تواند مستقیماً و بدون نیاز به دسترسی کرنل از آن استفاده کند. در مقابل، hard limit به‌عنوان یک آستانه نهایی و غیر قابل تجاوز تعریف می‌شود که فقط توسط کرنل و کاربر ریشه قابل تغییر است. رابطه بین این دو محدودیت به‌صورت $\text{hard limit} > \text{soft limit} > ۰$ تعریف می‌شود. سیستم‌عامل با توجه به این تنظیمات، دسترسی هر پردازه به منابعی مانند حافظه، تعداد فایل‌های باز و تعداد پردازه‌ها را کنترل می‌کند. اطلاعات مربوط به این محدودیت‌ها در مسیر /proc/ ذخیره می‌شود و کرنل با نظارت بر این فایل‌ها، میزان مصرف منابع توسط هر پردازه را مدیریت می‌کند [۱۲]. شکل ۲ محدودیت‌های مربوط به یک پردازه کارگر وب‌سرور Nginx با PID ۲۸۰ را نشان می‌دهد.

۲-۲ ایزوله‌سازی در کانتینرها

ایزوله‌سازی در کانتینرها بر پایه دو رویکرد اصلی انجام می‌گیرد که هر یک با اتخاذ استراتژی متفاوت، به دنبال دستیابی به تعادلی میان امنیت و کارایی هستند:

۱) استفاده از قابلیت‌های بومی سیستم‌عامل لینوکس: در این روش، کنترل منابع به‌طور مستقیم توسط ابزارهای درون‌ساختاری لینوکس مانند cgroups و فضای نام‌ها اعمال می‌شود. به عنوان نمونه (شکل ۳)، فناوری کانتینری LXC با بهره‌گیری از همین ابزارها، محدودیت‌های لازم را بر کانتینرها اعمال می‌کند. در این مدل، ابزار LXD که به‌عنوان یک مدیریت‌کننده سطح بالا برای LXC عمل می‌کند و رابط کاربری ساده‌تری برای پیکربندی و کنترل کانتینرها فراهم می‌سازد، با تخصیص منابع مشخص نظیر دو هسته پردازشی و چهار گیگابایت حافظه، فرایند ایزوله‌سازی را انجام می‌دهد. حذف واسطه‌هایی همچون دایمن، موجب کاهش سربار اجرایی شده و کنترل منابع را به‌صورت بهینه‌تری ممکن می‌سازد.

1. Daemon

```
root@ubuntu-nginx:~# cat /proc/cpuinfo | grep processor
processor       : 0
processor       : 1
root@ubuntu-nginx:~# free -m
               total        used        free      shared  buff/cache   available
Mem:            3814         139        3265           0         409        3675
Swap:              0           0           0
root@ubuntu-nginx:~# uname -a
Linux ubuntu-nginx 5.4.0-81-generic #91-Ubuntu SMP Thu Jul 15 19:09:17 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
root@ubuntu-nginx:~#
```

شکل ۳: محدودیت اعمال‌شده بر یک کانتینر LXC با استفاده از ویژگی‌های محدودسازی لینوکس.

```
root@3e09a5c4df03:/# cat /proc/cpuinfo | grep process | wc -l
32
root@3e09a5c4df03:/# cat /proc/meminfo | head -n 5
MemTotal:       16258024 kB
MemFree:         1630252 kB
MemAvailable:   12230608 kB
Buffers:          337552 kB
Cached:          9677040 kB
root@3e09a5c4df03:/# uname -a
Linux 3e09a5c4df03 5.4.0-81-generic #91-Ubuntu SMP Thu Jul 15 19:09:17 UTC 2021 x86_64 GNU/Linux
root@3e09a5c4df03:/#
```

شکل ۴: محدودیت اعمال‌شده بر یک کانتینر داکر با استفاده از دایمن.

```
root@e0abef298a7e:/# cat proc/cpuinfo | grep processor | wc -l
32
root@e0abef298a7e:/# cat proc/meminfo | head -n 5
MemTotal:       16258024 kB
MemFree:         1605304 kB
MemAvailable:   12217512 kB
Buffers:          339000 kB
Cached:          9685932 kB
root@e0abef298a7e:/# uname -a
Linux e0abef298a7e 5.4.0-81-generic #91-Ubuntu SMP Thu Jul 15 19:09:17 UTC 2021 x86_64 GNU/Linux
root@e0abef298a7e:/#
```

شکل ۵: محدودیت اعمال‌شده بر یک کانتینر پادمن با استفاده از دایمن.

۲) مدیریت فراخوانی‌های سیستمی از طریق دایمن: در این رویکرد، یک سرویس دایمن به‌عنوان واسطه‌ای بین کانتینر و کرنل عمل کرده و مسئول مدیریت درخواست‌های سیستمی برای دسترسی به منابع است. فناوری‌هایی نظیر داکر از این معماری بهره می‌برند؛ به این صورت که دایمن میزان مجاز استفاده از منابع را تعیین و کنترل می‌کند. با این حال، کانتینر همچنان امکان مشاهده کل منابع سیستم میزبان را دارد. این ویژگی ممکن است باعث شود که پردازه‌های درون کانتینر، تصویری نادرست از میزان واقعی منابع در دسترس دریافت کنند [۱۳].

رویکرد مبتنی بر دایمن این مزیت را دارد که امکان تغییر پیکربندی منابع را بدون نیاز به راه‌اندازی مجدد کانتینر فراهم کند. برای مثال در فناوری‌هایی مانند داکر و پادمن (مطابق شکل‌های ۴ و ۵)، کانتینرها قادرند مقادیر منابع تخصیص‌یافته را در لحظه مشاهده و متناسب با شرایط جدید به‌صورت پویا تنظیم کنند. این انعطاف‌پذیری، مدیریت منابع را در محیط‌های متغیر ساده‌تر می‌سازد. با این حال، استفاده از دایمن به‌عنوان واسطه در فرایند تخصیص و نظارت بر منابع می‌تواند در مقایسه با روش‌های مبتنی بر استفاده مستقیم از قابلیت‌های بومی لینوکس، سربار بیشتری به سیستم تحمیل کند.

در سال‌های اخیر، پژوهش‌های گسترده‌ای در زمینه فناوری کانتینرها، ابزارهای کانتینرسازی مانند داکر و کاربرد آنها در وب‌سرویس‌ها صورت گرفته است. کانتینرها در ابتدا با هدف استقرار کارآمد برنامه‌ها در محیط‌های محاسبات ابری توسعه یافتند و با جداسازی برنامه از وابستگی‌های آن، امکان سازگاری بیشتر و قابلیت حمل[1] بالاتری را فراهم کردند. ژو و همکارانش [۱۴]، استفاده از کانتینرها را در HPC[2] مورد بررسی قرار دادند و بر توان بالقوه این فناوری در ساده‌سازی فرایند استقرار برنامه‌ها از طریق ایجاد محیط‌های ایزوله که با الزامات امنیتی و عملکردی سخت‌گیرانه سازگار باشند، تأکید نمودند. با وجود پذیرش گسترده فناوری کانتینرسازی در محیط‌های ابری با ابزارهایی مانند داکر، پیاده‌سازی آن در سیستم‌های HPC با چالش‌هایی خاص همراه است؛ از جمله نیاز به کانتینرهای بزرگ‌تر، بهینه‌سازی برای سخت‌افزارهای تخصصی و محدودیت در ابزارهای هماهنگ‌سازی[3]. نویسندگان در این راستا به بررسی موتورهای کانتینری تخصصی برای HPC مانند Shifter و Singularity پرداخته‌اند و ویژگی‌هایی نظیر اجرای بدون نیاز به دسترسی ریشه[4]، پشتیبانی از MPI[5] و سازگاری با GPU را مورد تحلیل قرار داده‌اند. یافته‌های آنها نشان می‌دهد که اگرچه این راه‌حل‌ها عملکرد نزدیک به سیستم‌های بومی و سطح بالایی از انعطاف‌پذیری را فراهم می‌کنند، همچنان در زمینه‌هایی نظیر ناسازگاری بین پلتفرم‌ها، چالش‌های امنیتی و محدودیت در هماهنگی منابع با موانعی روبه‌رو هستند.

سینگ و همکارانش [۱۵] یک سامانه توازن بار و کشف سرویس مبتنی بر Docker Swarm را برای برنامه‌های کلان‌داده مبتنی بر معماری مایکروسرویس پیشنهاد کردند. این پژوهش به بررسی چالش‌های موجود در مدیریت پردازش داده‌های عظیم با استفاده از کانتینرهای داکر می‌پردازد و بر بهبود ویژگی‌هایی مانند انعطاف‌پذیری، مقیاس‌پذیری و سهولت استقرار متمرکز است. در این رویکرد از Docker Swarm

1. Portability
2. High Performance Computing
3. Orchestration
4. Non-Root Execution
5. Message Passing Interface

به‌عنوان ابزار هماهنگ‌سازی جهت مدیریت و توزیع پویا بارهای کاری در میان چندین گره استفاده شده است. این سامانه با پایش مصرف حافظه در هر گره، بهره‌وری در استفاده از منابع را تضمین می‌کند. معماری پیشنهادی بر روی یک برنامه کلان‌داده با بهره‌گیری از پشته مایکروسرویس پیاده‌سازی و ارزیابی شده است. نتایج نشان می‌دهند که سامانه قادر است بار کاری را به‌طور مؤثر توزیع نماید، عملکرد کلی را ارتقا دهد و از یکپارچگی سیستم پشتیبانی کند. آرزو و همکارانش [۱۶] چارچوب جامعی را برای ارزیابی معماری‌های مبتنی بر مایکروسرویس‌ها برای شبکه‌های ۶G آینده با تأکید بر KPIهای اصلی مانند تأخیر، توان عملیاتی و مصرف انرژی پیشنهاد کردند. این رویکرد منابع شبکه را با استفاده از یک هایپرگراف چندلایه برای ترسیم تعاملات در لایه‌های فیزیکی و مجازی، از جمله محیط‌های لبه و ابری مدل‌سازی می‌کند. این چارچوب فرمول‌های ریاضی را برای ثبت محدودیت‌های تأخیر و استفاده از منابع و انرژی با در نظر گرفتن اثرات کانتینری‌سازی مانند تأخیرهای مجازی‌سازی، ترکیب می‌کند. اعتبارسنجی تجربی با چارچوب SDN افزایش مصرف توان داکر را نشان داد، اما مزایای مقیاس‌پذیری آن را برجسته کرد. در [۱۷] با تمرکز بر ارزیابی عملکرد فناوری‌های مختلف کانتینر مانند RunC، LXC، Containerd، Docker، Podman و Singularity در برنامه‌های بینایی کامپیوتر مبتنی بر OpenCV بر روی دستگاه‌های لبه مبتنی بر ARM تحقیقاتی انجام شده است. این مقاله به شکاف در تحقیقات مربوط به عملکرد کانتینر در حوزه‌های خاص، به ویژه در محاسبات لبه برای بینایی رایانه می‌پردازد. این نشان می‌دهد که برنامه‌های کانتینری علی‌رغم برخی تغییرات در عملکرد، نتایج قابل مقایسه با برنامه‌های غیرکانتینری را نشان می‌دهند. قابل ذکر است که Containerd در خواندن حافظه و دریافت تصویر عملکرد خوبی دارد، در حالی که داکر اگرچه در خواندن حافظه کندتر است، اما در زمان پردازش از دیگران بهتر است. در این پژوهش سه سناریو پیاده‌سازی بررسی شده است. سناریوی اول شامل برنامه‌ای است که تصاویر را از حافظه می‌خواند، جایی که هم برنامه و هم تصاویر در یک کانتینر قرار دارند. سناریوهای دوم و سوم بر اساس نوع اتصال متفاوت هستند: به ترتیب با سیم و بی‌سیم. با توجه به نتایج، در حالی که زمان‌های اجرا کانتینر مانند Docker، Singularity و Podman عملکرد قوی را نشان می‌دهند، اجرای بومی روی RaspberryPi ۴ به طور مداوم از نظر زمان پردازش از کانتینرها بهتر عمل می‌کند، به ویژه در سناریوهایی که شامل استفاده از منابع بالا یا محاسبات پیچیده است. این نشان می‌دهد که برای برنامه‌های بلادرنگ در دستگاه‌های با محدودیت منابع مانند Raspberry Pi ۴، اجرای بومی ممکن است انتخاب بهینه‌تری برای برخی از وظایف بینایی رایانه باشد. با این حال، کانتینرها هنوز هم از نظر مقیاس‌پذیری و انعطاف‌پذیری استقرار مزایای قابل توجهی دارند و آنها را در سیستم‌های بزرگ‌تر یا توزیع‌شده ارزشمند می‌کند. سیلوا و همکارانش [۱۸] مطالعه تطبیقی از فناوری‌های کانتینر Docker و LXD ارائه و محدودیت‌ها و مزایای کانتینرهای کاربردی (Docker) در مقابل سیستم (LXD) را مشخص کردند. این مطالعه با استفاده از معیارهای بین CPU، دیسک و شبکه نشان داد که هر دو کانتینر در مقایسه با محیط‌های بومی با حداقل هزینه سربار کارآمدی دارند. سیستم فایل لایه‌ای داکر قابلیت حمل و استقرار در مایکروسرویس‌ها را افزایش می‌دهد، در حالی که LXD پایداری را فراهم می‌کند و از مجازی‌سازی کامل سیستم‌عامل پشتیبانی می‌کند. به طور کلی، LXD به پایداری بالاتری در وظایف با منابع فشرده دست یافت و آن را به عنوان یک جایگزین بالغ در مجازی‌سازی سیستم قرار داد. در [۱۹] به ارزیابی عملکرد فناوری‌های مجازی‌سازی نظیر VirtualBox، VMware، QEMU و نیز کانتینری‌سازی (Docker، Podman و LXD) که شامل microVMsهایی نظیر Firecracker و QEMU می‌شود، پرداخته شده است. محققین این پژوهش با استفاده از معیار SysBench، عملکرد CPU، RAM و دیسک را ارزیابی کردند. نتایج نشان داد که راه‌حل‌های کانتینری‌سازی عموماً از مجازی‌سازی در کارایی CPU و RAM بهتر عمل می‌کنند و LXD و Podman بهترین نتایج را نشان می‌دهند. MicroVMها عملکرد رقابتی نزدیک به کانتینرها را ارائه می‌دادند، اما فاقد ابزار پشتیبانی بودند. VMware در عملیات دیسک برتر بود، در حالی که Firecracker در سرعت خواندن/ نوشتن پیشرو بود که نشان‌دهنده مناسب‌بودن هر فناوری برای وظایف خاص است. دوی و همکارانش [۲۰] نیز یک مطالعه دقیق در مورد امنیت کانتینر، تجزیه و تحلیل سطوح تهدید، استراتژی‌های کاهش ریسک و زمینه‌های تحقیقات آینده را بررسی کردند. آنها از مدل‌سازی DREAD برای اولویت‌بندی ریسک‌ها در طول چرخه عمر کانتینر استفاده نمودند و کانتینرهایی با مجوز کاربر روت/ بدون روت را مورد آزمایش قرار دادند. این مطالعه شامل یک مدل تهدید سیستماتیک، سوء استفاده‌های واقعی و مقایسه ابزارهای رایج ارزیابی آسیب‌پذیری است.

۳- مدل پیشنهادی

در این بخش مدل ارائه‌شده در خصوص معماری کانتینرها مبتنی بر وب‌سرورهای مختلف مورد بررسی قرار می‌گیرد.

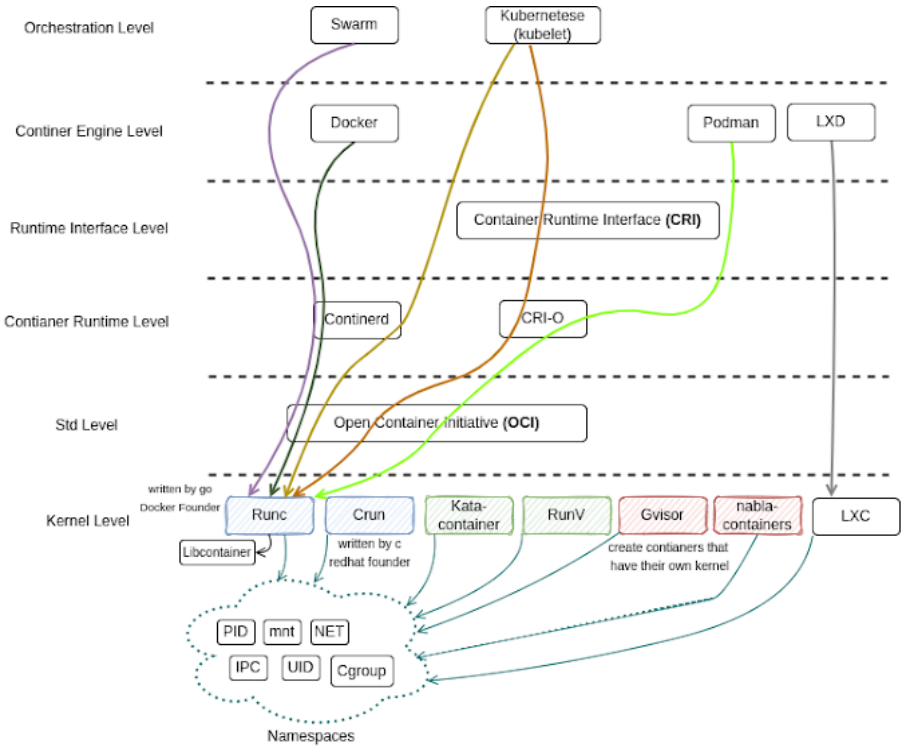
۳-۱ مدل لایه‌ای کانتینرها

به‌منظور درک دقیق‌تر معماری فناوری‌های کانتینرسازی، این مقاله یک مدل مفهومی شش‌لایه‌ای را معرفی می‌کند. در این مدل، هر لایه نقش مشخصی در بهینه‌سازی عملکرد کانتینرها و مدیریت مؤثر منابع ایفا می‌کند. لایه‌ها از سطح Orchestration که مسئول مدیریت خودکار و مقیاس‌پذیری کانتینرهاست، آغاز شده و تا لایه کرنل که شامل فراخوانی‌های سیستمی و تخصیص منابع سخت‌افزاری است، ادامه می‌یابند. همان طور که در شکل ۶ نمایش داده شده، مسیرهای ارتباطی و فراخوانی‌ها بین لایه‌ها با فلش مشخص شده‌اند و این ساختار سلسله‌مراتبی به تحلیل دقیق‌تر تعاملات میان مؤلفه‌های مختلف کمک می‌کند. مدل شش‌لایه‌ای، ضمن تفکیک وظایف در هر سطح، نحوه عملکرد فناوری‌هایی مانند داکر، پادمن و LXC را در مدیریت و اجرای کانتینرها به‌خوبی تبیین می‌کند. علاوه بر این، چارچوب ارائه‌شده امکان مقایسه معماری این فناوری‌ها، شناسایی نقاط قوت و ضعف هر یک و بررسی میزان سازگاری آنها با وب‌سرورهایی نظیر Apache و Nginx را فراهم می‌سازد. در مجموع، بهره‌گیری از این مدل در پژوهش حاضر، نه‌تنها به تشریح فنی ساختار لایه‌ای کانتینرها کمک می‌کند، بلکه بستری مفهومی برای درک رابطه بین منابع سیستم، فناوری کانتینری و عملکرد نرم‌افزارهای کاربردی فراهم می‌سازد.

۳-۱-۱ Orchestration Level

این لایه، سرویس‌هایی مانند مقیاس‌پذیری، زمان‌بندی، پایش سلامت کانتینر و مدیریت بار کاری[1] را برای مدیریت بهینه کانتینرها فراهم می‌کند. یکی از شناخته‌شده‌ترین ابزارهای این لایه، Kubernetes (K۸s)

1. Workload



شکل ۶ مدل شش‌لایه معماری مجازی‌سازی مبتنی بر کانتینر.

است که به عنوان معروف‌ترین ابزار مدیریت هماهنگ‌سازی در جهان شناخته می‌شود. Kubernetes در Google Cloud ایجاد شد و در سال ۲۰۱۴ به عنوان یک پروژه متن‌باز معرفی گردید. این ابزار که اکنون توسط بنیاد رایانش ابری (CNCF) توسعه و نگهداری می‌شود، امکان مقیاس‌پذیری، استقرار و مدیریت خودکار برنامه‌های حاوی کانتینرها را فراهم می‌آورد [۲۱]. از دیگر ابزارهای محبوب مدیریت هماهنگ‌سازی در این لایه، داکر Swarm است که به طور خاص برای کنترل و مدیریت کانتینرهای داکر طراحی شده است. داکر Swarm به برنامه‌ها امکان می‌دهد تا در چندین گره به‌صورت یکپارچه و همگام با یکدیگر اجرا شوند [۱۵]. این ابزار به‌منظور مدیریت و مقیاس‌پذیری کلاسترهای داکر طراحی شده و از ورودی‌های خط فرمان یا فایل‌های پیکربندی YAML و JSON برای تنظیم و ساخت کانتینرها استفاده می‌کند و بسته به نوع فناوری از لایه‌های Runtime Interface و Container Runtime کمک می‌گیرد.

Container Engine Level ۳-۱-۲

در این لایه، ابزارهای اصلی برای تعامل کاربران با فناوری‌های کانتینری ارائه می‌شوند. از شناخته‌شده‌ترین ابزارهای این سطح می‌توان به داکر، پادمن و LXD اشاره کرد. در این لایه، دستورات و پیکربندی‌ها معمولاً از طریق واسط خط فرمان دریافت شده و سپس با فراخوانی ماژول‌های لایه Container Runtime، فرایند ایجاد و اجرای کانتینرها آغاز می‌شود. ابزار LXD با بهره‌گیری مستقیم از ماژول‌های کرنل لینوکس، عملکردی نزدیک به سطح سیستم‌عامل دارد. در مقابل، پادمن مبتنی بر استاندارد OCI[1] عمل می‌کند که توسط پروژه Kubernetes توسعه یافته است. یکی از ویژگی‌های مهم پادمن، امکان اجرای کانتینرها بدون نیاز به دایمن است؛ به این معنا که فرایند ساخت و اجرای کانتینرها بدون وابستگی به سرویسی با دسترسی سطح ریشه[2] انجام می‌شود که این امر موجب افزایش سطح امنیت در تعامل بین کانتینرها و سیستم میزبان می‌گردد. همچنین فناوری LXC به‌عنوان یکی از روش‌های مجازی‌سازی در سطح سیستم‌عامل شناخته می‌شود که امکان اجرای سیستم‌های لینوکسی ایزوله‌شده را بر روی یک میزبان مشترک فراهم می‌سازد. LXC از هسته لینوکس به‌طور مستقیم استفاده می‌کند و با بهره‌گیری از Cgroup، قابلیت کنترل و اولویت‌بندی منابعی نظیر پردازنده، حافظه و سایر منابع سیستم را فراهم می‌آورد؛ بدون آنکه نیاز به پیکربندی مستقیم فضای نام‌ها باشد. در میان فناوری‌های کانتینری، LXC به‌دلیل سطح بالای ایزوله‌سازی و شباهت ساختاری به ماشین‌های مجازی، به‌ویژه در سناریوهای امنیت‌محور از محبوبیت قابل توجهی برخوردار است [۱۸].

Runtime Interface Level ۳-۱-۳

برای امکان‌پذیرساختن استفاده از فناوری‌های کانتینرسازی توسط سایر برنامه‌ها، وجود یک واسط استاندارد میان لایه‌های بالادستی و موتور اجرای کانتینرها ضروری است؛ لایه "Runtime Interface" به این نیاز پاسخ می‌دهد. در این لایه، مهم‌ترین سرویس، CRI[3] است که به کاربران و یا ابزارهای مدیریت خوشه مانند Kubernetes، اجازه می‌دهد که پیکربندی‌های مورد نیاز برای ساخت و اجرای کانتینر را در قالب فایل‌هایی با فرمت JSON تعریف کنند؛ بدون آنکه مستقیماً به فراخوانی سرویس‌های لایه موتور زمان اجرای کانتینر[4] نیاز باشد. یکی از کلیدی‌ترین سرویس‌هایی که از این واسط بهره می‌برد، kubelet است؛ مؤلفه‌ای در معماری Kubernetes که مسئولیت مدیریت، هماهنگی و اجرای کانتینرها را با استفاده از CRI بر عهده دارد [۲۲]. این لایه،

1. Open Container Initiative

2. Root
3. Container Runtime Interface
4. Container Runtime Engine

پیکربندی‌ها و دستورات دریافتی از لایه‌های بالا مانند Orchestration را پردازش می‌نماید و به‌عنوان واسطه، آنها را برای اجرای کانتینر به لایه پایین‌دستی یعنی زمان اجرای کانتینر منتقل می‌کند. طراحی این لایه در چارچوب پروژه Kubernetes، به‌ویژه با در نظر گرفتن نیازهای ارائه‌دهندگان خدمات ابری، مبتنی بر استاندارد OCI صورت گرفته است تا از سازگاری و انعطاف‌پذیری بالاتر در محیط‌های متنوع اطمینان حاصل شود.

۳-۱-۴ Container Runtime Level

در لایه Container Runtime، کلیه مراحل مرتبط با چرخه حیات کانتینرها مدیریت می‌شود؛ از جابه‌جایی Imageها و تخصیص فضای ذخیره‌سازی گرفته تا اجرای کانتینر و کنترل منابع مرتبط. دو نمونه از مهم‌ترین Runtimeهای کانتینری در این لایه عبارتند از Containerd و CRI-O. این Runtimeها تنظیمات و فایل‌های پیکربندی دریافتی از لایه‌های بالادستی را دریافت می‌نمایند و پس از پردازش، دستورات لازم را از طریق واسط استاندارد به کرنل سیستم‌عامل ارسال می‌کنند. Containerd که در ابتدا توسط تیم توسعه داکر ارائه شد، به‌عنوان یک Runtime عمومی و پایدار برای اجرای کانتینرها شناخته می‌شود. این ابزار مدت‌ها توسط Kubernetes نیز مورد استفاده قرار می‌گرفت تا اینکه CRI-O به‌عنوان Runtime اختصاصی این پلتفرم معرفی شد. با وجود این، Kubernetes همچنان پشتیبانی از Containerd را ادامه می‌دهد و داکر نیز از این Runtime برای انجام عملیات‌هایی مانند اجرا، توقف یا تغییر کانتینرها بهره می‌برد. CRI-O که به‌طور ویژه برای هماهنگی با Kubernetes طراحی شده، به‌عنوان یک سرویس مستقل و سبک‌وزن عمل می‌کند که بدون نیاز به دایمن و دسترسی سطح ریشه فعالیت می‌نماید. این ویژگی موجب کاهش سربار اجرایی و افزایش امنیت در سیستم می‌شود. در مقابل، Containerd مبتنی بر مدل دایمن است؛ به این معنا که اجرای کانتینرها از طریق یک فرایند زمینه‌ای صورت می‌گیرد. با ظهور فناوری پادمن، امکان بهره‌گیری مستقیم از CRI-O بدون نیاز به ابزارهای هماهنگ‌سازی مانند Kubernetes نیز فراهم شد که این موضوع انعطاف‌پذیری بیشتری برای پیاده‌سازی سبک و امن کانتینرها ایجاد می‌کند [۲۳].

۳-۱-۵ STD Level

لایه استاندارد یک لایه تجریدی است که نقش مهمی در تسهیل ارتباط بین لایه‌های بالاتر و ماژول‌های کرنلی ایفا می‌کند. این لایه بدون نیاز به درگیرشدن با پیچیدگی‌های ماژول‌های کرنلی، مراحل ساخت کانتینر را انجام می‌دهد. یکی از مهم‌ترین استانداردهای این لایه OCI است که توسط تمامی فناوری‌های کانتینرسازی از جمله داکر و پادمن برای فراخوانی‌های لایه کرنلی استفاده می‌شود [۲۴]. در واقع، لایه STD تمام اطلاعات ضروری برای ساخت کانتینر (مانند دستورات خط فرمان، نسخه کرنل و نوع برنامه) را از لایه زمان اجرای کانتینر دریافت می‌کند و به عنوان تنها لایه استاندارد، این اطلاعات را به کرنل ارسال می‌کند تا روند ایجاد کانتینر آغاز شود.

۳-۱-۶ Kernel Level

در لایه کرنل، ماژول‌های هسته سیستم‌عامل با بهره‌گیری از فناوری‌هایی مانند فضای نام‌ها و Cgroup، کانتینرها را ایجاد می‌کند و منابع سخت‌افزاری و نرم‌افزاری مورد نیاز را به آنها تخصیص می‌دهند. این لایه مسئول تفسیر دستورات دریافتی و تعامل مستقیم با سخت‌افزار است و نقشی حیاتی در مدیریت منابع سیستم ایفا می‌کند. ماژول‌های مورد استفاده در این لایه برای پشتیبانی از مجازی‌سازی کانتینری در سه دسته کلی طبقه‌بندی می‌شوند [۹]، [۲۳] و [۲۵]:

- Native Runtimes: در این دسته، کانتینرها مستقیماً بر روی کرنل میزبان اجرا می‌شوند. این روش از سرعت بالاتری نسبت به سایر مدل‌ها برخوردار است، اما سطح ایزوله‌سازی و امنیت آن پایین‌تر است. از مهم‌ترین نمونه‌های این نوع می‌توان به runc و crun اشاره کرد که هر دو مطابق با استاندارد OCI توسعه یافته‌اند.
  - runc توسط تیم داکر با زبان Go توسعه یافته و به‌طور گسترده در ابزارهایی نظیر داکر، پادمن و CRI-O مورد استفاده قرار می‌گیرد.
  - crun نیز توسط شرکت Red Hat و با زبان C توسعه داده شده و به دلیل مصرف کمتر منابع و سرعت بالاتر، در برخی سناریوها ترجیح داده می‌شود.
- Sandbox Runtimes: این نوع ران‌تایم‌ها برای افزایش امنیت از یک لایه پروکسی کرنل برای ارتباط با کرنل میزبان بهره می‌برند و محیطی ایزوله‌تر ایجاد می‌کنند. این ساختار، نفوذ به سطح سیستم‌عامل را دشوارتر می‌کند و در سناریوهای امنیت‌محور کاربرد دارد. از شناخته‌شده‌ترین نمونه‌ها در این دسته می‌توان به RunV و Kata Containers اشاره کرد. این مدل پیش‌تر در محیط‌هایی با الزامات امنیتی بالا برای کاهش سطح تهدید به کار گرفته می‌شد.
- Virtualized Runtime: در این رویکرد، هر کانتینر در قالب یک ماشین مجازی کامل و مجزا اجرا می‌شود که دارای کرنل اختصاصی خود است. این مدل بیشترین میزان ایزوله‌سازی را فراهم می‌کند و برای مواردی با نیازهای امنیتی خاص و شدید مناسب است. از جمله ابزارهای مطرح در این دسته می‌توان به gVisor و Nabla-Container اشاره کرد. این ابزارها با ارائه لایه مجازی‌سازی عمیق‌تر، امکان جداسازی کامل پردازه‌ها را از یکدیگر فراهم می‌سازند [۲۶] و [۲۷].

۳-۲ دسته‌بندی برنامه‌ها

برنامه‌ها به‌طور کلی بر اساس میزان و نوع استفاده از منابع سیستم به چهار دسته تقسیم می‌شوند. هر برنامه می‌تواند بسته به کاربردش به یک یا چند دسته از این گروه‌ها تعلق داشته باشد [۲۸]:

۱) برنامه‌های I/Oمحور: این برنامه‌ها بیشترین بار کاری را بر منابع ذخیره‌سازی وارد می‌کنند. از جمله این برنامه‌ها می‌توان به پایگاه‌های داده و نرم‌افزارهایی که به سیستم فایل وابسته هستند، اشاره کرد.

۲) برنامه‌های حافظه‌محور: این دسته از برنامه‌ها بیشترین مصرف منابع را در حافظه به خود اختصاص می‌دهند. نرم‌افزارهای مدیریت صف و حافظه نهان مانند Redis و Kafka، نمونه‌هایی از این برنامه‌ها هستند.

۳) برنامه‌های پردازش‌محور: این برنامه‌ها نیازمند پردازش سنگین هستند و به منابع پردازشی زیادی احتیاج دارند. از جمله آنها می‌توان به نرم‌افزارهای فشرده‌سازی و رمزگذاری اشاره کرد.

۴) برنامه‌های شبکه‌محور: این برنامه‌ها بیشتر از کانال‌های ارتباطی استفاده می‌کنند. مرورگرها، نرم‌افزارهای دانلود و وب‌سرورها در این دسته قرار دارند.

برنامه‌ای مانند Nginx به عنوان یک وب‌سرور، کش‌سرور و پروکسی سرور را می‌توان همزمان عضو سه دسته مقید به حافظه، شبکه و CPU در نظر گرفت و یا برنامه‌هایی مانند پایگاه داده mysql را می‌توان در سه

دسته مقید به حافظه، I/O و CPU لحاظ نمود. این دسته‌بندی به انتخاب و تنظیم صحیح منابع برای هر نوع برنامه کمک می‌کند و امکان استفاده بهینه از منابع سیستم را فراهم می‌آورد [۲۹].

۳-۳ وب‌سرورها

بزرگ‌ترین چالش موجود بین وب‌سرورها همروندی[1] است. همروندی در وب‌سرورها به قابلیت پاسخ همزمان به چندین ارتباط اطلاق می‌شود. در گذر زمان و افزایش سرعت اینترنت و ظهور شبکه‌های موبایل، معماری وب‌سرورها نیز دستخوش تغییرات عمده‌ای شده است [۳۰] و [۳۱]. در ابتدا با ظهور وب‌سرور Apache در دهه نود، این وب‌سرور به عنوان یک تک‌برنامه که به‌صورت stand alone اجرا می‌شد شروع به کار کرد. دهه ابتدایی قرن بیستم، معماری یک پردازه به ازای هر یک درخواست رایج شد که این امر برای مقیاس‌های بزرگ مناسب نبود. این مشکل با وب‌سرور مبتنی بر ارتباط بهبود پیدا کرد؛ بدین صورت که به ازای هر درخواست، یک ارتباط با برنامه Apache برقرار می‌شد. با این کار مشکل مقیاس‌پذیری بهبود یافت، ولی به ازای هر ارتباط، یک مگابایت از فضای حافظه اشغال می‌شد. این چالش با ظهور ارتباط پایدار[2] حل شد. همچنین با این راه‌حل، تأخیر ناشی از توافق‌های اولیه ناشی از ایجاد ارتباط (دست‌تکانی سه‌مرحله‌ای) به ازای هر درخواست کاهش پیدا کرد. بعد از مدتی، چالش C۱۰K توسط مهندس نرم‌افزاری با نام Daniel Kagel مطرح شد که طبق این نگرش به سرورهای وب اجازه سرویس به بیشتر از ده هزار کاربر همزمان را نمی‌داد. با مهاجرت وب‌سرورها از مدل مبتنی بر نخ به مدل مبتنی بر رخداد مانند Nginx این چالش نیز حل شد. در ابتدا قرار بود پروژه Nginx به همروندی Apache به عنوان وب‌سرور کمک کند، اما با پشتیبانی از پروتکل‌های SCGI و Uswgi FastCGI و اضافه‌کردن سیستم کش توزیع‌شده[3] و تعریف توابعی برای تبدیل‌کردن این پروژه به یک Reverse-proxy Load-balancer، پروژه Nginx به یک وب‌سرور چندمنظوره تبدیل شد [۳۲] و [۳۳]. در گذشته برای ساخت وب‌سرویس‌ها، استفاده از معماری LAMP[4] بسیار چشمگیر بود. در این معماری، لینوکس به‌عنوان سیستم‌عامل، Apache به عنوان وب‌سرور، Mysql به منظور پایگاه داده و PHP به عنوان زبان مفسری به کار می‌رود، اما امروزه با توجه به پشتیبانی Nginx از SSL، حافظه نهان، فشرده‌سازی و بهبودهای همزمانی، معماری LEMP[5] مورد توجه قرار گرفت که در آن Nginx جایگزین Apache شده است [۳۴]. وب‌سرورها را از لحاظ معماری می‌توان در دو گروه دسته‌بندی نمود [۳۵]:

۱) وب‌سرورهای Process-based و Thread-based: در این نوع وب‌سرورها، هر ارتباط توسط یک پردازه یا نخ جداگانه مدیریت می‌شود که ممکن است به استفاده غیربهینه از منابع منجر شود. به دلیل نیاز به اختصاص حافظه و محیط‌های جداگانه برای هر ارتباط، با افزایش تعداد درخواست‌ها و پردازه‌ها، سربار سیستم بیشتر می‌شود و عملکرد کاهش می‌یابد. آپاچی از این روش برای مدیریت درخواست‌ها استفاده می‌کرد.

۲) وب‌سرورهای Event-based: Nginx از ابتدای کار به استفاده اقتصادی از منابع مشهور بوده است. این وب‌سرور به کمک مدل مبتنی بر رویداد، ارتباط‌ها را به صورت Asynchronous و Non-Blocking پردازش می‌کند. درخواست‌ها با استفاده از Multiplexing و Event Notification در یک حلقه رویداد به پردازه‌های تک‌نخی یا Workerها تخصیص داده می‌شوند که هر کدام می‌توانند همزمان به هزاران ارتباط پاسخ دهند.

از مهم‌ترین ویژگی‌های Nginx می‌توان به موارد زیر اشاره کرد:

- Non-Blocking: در این حالت به جای انتظار برای تکمیل منابع درخواست فعلی، وب‌سرور Nginx درخواست‌ها را به حلقه رویداد بازمی‌گرداند تا درخواست‌های بعدی پردازش شوند. این ویژگی باعث می‌شود Nginx بتواند همزمان به هزاران ارتباط پاسخ دهد.
- Single-Thread: برخلاف معماری‌های سنتی چندپردازه‌ای یا چندنخی، در Nginx تمامی امور توسط Workerهای تک‌نخی انجام می‌شود که تعویض متن را کاهش داده و بهبود کارایی را به همراه دارد.

در Nginx، پردازه Master وظیفه خواندن فایل‌های پیکربندی و مدیریت سوکت‌ها و Workerها را بر عهده دارد. Workerها به عنوان فرایندهای فرزند توسط ویژگی‌های Fork یا Clone لینوکس ایجاد می‌شوند و درخواست‌های جدید را از طریق یک حلقه رویداد پردازش می‌کنند. این معماری، ترکیبی از Non-Blocking و Event-loop است که امکان پردازش درخواست‌های بی‌شمار را فراهم می‌کند [۳۶].

۳-۴ ارزیابی عملکرد وب‌سرورها بر بستر کانتینرها

برای ارزیابی عملکرد وب‌سرورهای Apache و Nginx در محیط‌های کانتینری مبتنی بر داکر، پادمن و LXC، سناریوهای مختلفی با محدودیت‌های سخت‌افزاری گوناگون در نظر گرفته شده است. ابزار Apache Benchmark به عنوان ابزار اصلی ارزیابی به کار گرفته شده که معیار سنجش آن بر اساس تعداد ارتباط‌های همزمان یا همروندی تعریف شده است. در این آزمایش، تمامی کانتینرها بر اساس سطح منابع به سه دسته تقسیم شده‌اند. انتخاب سطوح منابع در سه سناریوی پایه، متوسط و بدون محدودیت بر اساس الگوهای رایج در استقرار واقعی برنامه‌های کانتینری انجام شده است:

۱) سطح اول (پیکربندی پایه): در این سطح، تمامی کانتینرها دارای ۲ هسته پردازشی CPU و ۴ گیگابایت حافظه هستند. این پیکربندی یکی از متداول‌ترین تنظیمات در معماری‌های مایکروسرویس است که برای حفظ ثبات عملکردی برنامه‌ها استفاده می‌شود. بیشتر فناوری‌های مدیریت هماهنگ‌سازی مانند Kubernetes از این پیکربندی پایه به عنوان حداقل منابع لازم برای اجرای پایدار برنامه‌های مبتنی بر کانتینر بهره می‌برند.

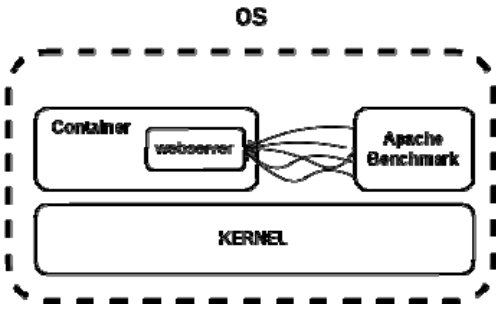
۲) سطح دوم (پیکربندی متوسط): در این پیکربندی به هر کانتینر، ۴ هسته CPU و ۸ گیگابایت حافظه اختصاص داده شده است. این تنظیم برای مواردی مناسب است که برنامه‌ها نیاز به منابع بیشتری دارند؛ به‌ویژه در مهاجرت از ماشین‌های مجازی به کانتینرها. استفاده از منابع در مقادیر توان ۲ (مانند ۲، ۴، ۸ و غیره) باعث جلوگیری از قطعه‌قطعه‌شدن حافظه و استفاده بهینه از منابع سیستم‌عامل می‌شود.

۳) سطح سوم (پیکربندی بدون محدودیت): در این سطح، کانتینرها به تمام منابع سخت‌افزاری سیستم میزبان، یعنی منابع Bare Metal، دسترسی دارند و محدودیتی برای استفاده از CPU و حافظه اعمال نشده است. این پیکربندی عموماً در محیط‌های مایکروسرویس

1. Concurrency
2. Persistent Connection
3. Memcache
4. Linux+Apache+Mysql+Php
5. Linux+Nginx+Mysql+Php

```
umz@umz-uni:~$ sudo dmidecode | grep -A3 '^System Information'
System Information
        Manufacturer: HP
        Product Name: ProLiant DL380 Gen9
        Version: Not Specified
umz@umz-uni:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal
umz@umz-uni:~$ free -m
               total        used        free      shared  buff/cache   available
Mem:           15876        3027        2546           2       10303       12517
Swap:          14335          15       14320
umz@umz-uni:~$ nproc
32
umz@umz-uni:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          32
On-line CPU(s) list:             0-31
Thread(s) per core:              2
Core(s) per socket:              8
Socket(s):                       2
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           79
Model name:                      Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
```

شکل ۷: مشخصات محیط پیاده‌سازی.



شکل ۸: معماری درونی محیط آزمون.

مرسوم نیست و فقط برای ارزیابی و مقایسه عملکرد در شرایط مختلف در نظر گرفته شده است.

۳-۵ محیط پیاده‌سازی آزمایش

طبق شکل ۷، آزمایش‌ها روی یک سرور HP نسل ۹ با سیستم‌عامل Ubuntu Server ۲۰.۰۴ LTS انجام شده که دارای ۱۶ گیگابایت حافظه از نوع DDR۴ و پردازنده Intel Xeon E۵-۲۶۲۰ v۴ با ۳۲ هسته مجزا و پشتیبانی از KVM است.

۳-۶ فرایند اجرای آزمون‌ها

در این سناریوها، عملکرد دو وب‌سرور Apache و Nginx بر بستر سه فناوری کانتینرسازی معروف داکر، پادمن و LXC بررسی خواهد شد. ابزار Apache Benchmark به منظور بررسی میزان عملکرد وب‌سرورها استفاده شده است. در تمامی آزمون‌ها، میزان همروندی، پارامتر ورودی است که مبنای بررسی عملکرد نیز واقع شده است. همروندی، تعداد ارتباط‌های بین کانتینر و برنامه Apache Benchmark (که بر روی هاست اصلی در حال اجرا است) را که در وضعیت ESTABLISH قرار دارند، نشان می‌دهد. به بیانی دیگر، این ابزار آزمون روی سیستم‌عامل میزبان نصب شده و با ارسال درخواست به وب‌سروری که بر روی کانتینر موجود است، اقدام به ارزیابی عملکرد وب‌سرورها بر اساس سرعت پاسخگویی به درخواست‌ها می‌کند. شکل ۸ شماتیک این معماری ارزیابی را نشان می‌دهد.

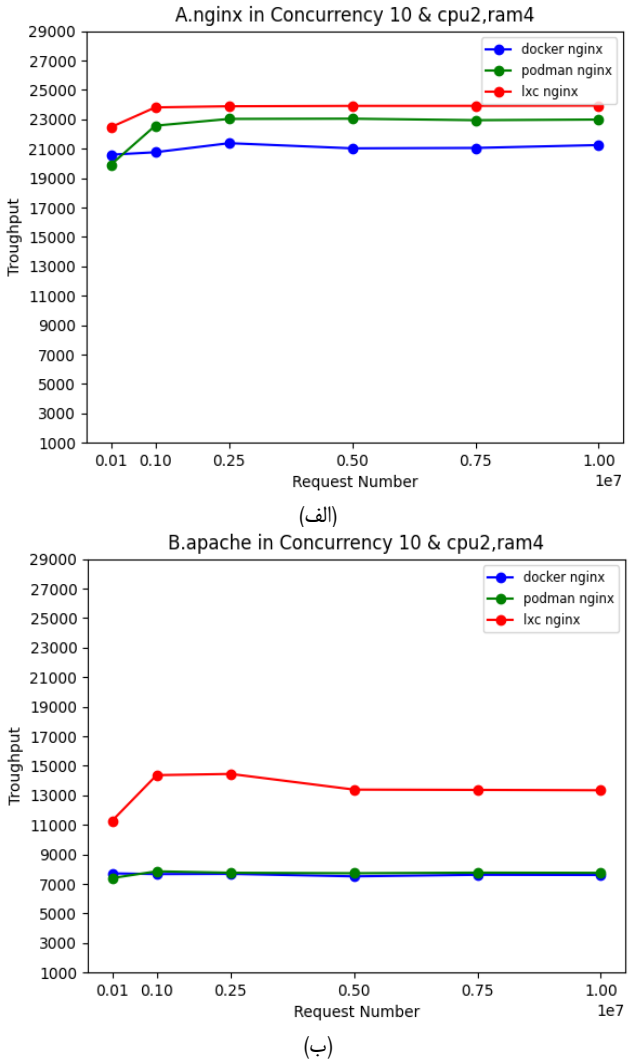
۳-۷ معیارهای ارزیابی

همروندی یکی از مهم‌ترین شاخص‌های ارزیابی وب‌سرورها است و نظریه C۱۰K نیز بر اساس این معیار شکل گرفته است. این شاخص در سناریوهای واقعی به تعداد کاربران همزمانی که وب‌سرور قادر به پاسخگویی به آنها است، اشاره دارد. نتایج هر آزمون در نمودارهایی ارائه شده که محور افقی نشان‌دهنده تعداد درخواست‌ها و محور عمودی بیانگر نرخ گذردهی[1] است. نرخ گذردهی در این آزمون، تعداد درخواست‌هایی است که در هر ثانیه بین ابزار Apache Benchmark و کانتینر رد و بدل می‌شود. به عنوان مثال در شکل ۹، ارسال تعداد درخواست $۰/۱\times۱۰^{۷}$ برای Nginx که برای روی کانتینر LXC قرار گرفته است با همروندی ۱۰ درخواست، نرخ گذردهی برابر با ۲۳۰۰۰ است. یعنی اگر این تعداد درخواست به سمت سرور ارسال شود و در هر زمان امکان ایجادشدن ۱۰ ارتباط همزمان میسر باشد، تعداد ۲۳۰۰۰ پاسخ در ثانیه دریافت خواهد شد.

لازم به ذکر است معیارهای منتخب برای ارزیابی شامل نرخ گذردهی و تعداد ارتباط‌های همزمان (همروندی) به دلایل زیر انتخاب شده‌اند:

- این معیارها مستقیماً با عملکرد وب‌سرورها در ارتباط هستند و نشان می‌دهند سیستم زیر چه میزان فشار کاری، چه نرخ پاسخ‌دهی دارد.
- معیار همزمانی به‌طور خاص با چالش C۱۰K مرتبط است که به‌طور تاریخی در ارزیابی وب‌سرورها استفاده می‌شود.
- ابزار Apache Benchmark به عنوان ابزار رسمی و استاندارد در صنعت، این معیارها را پشتیبانی می‌کند.

1. Throughput



شکل ۹: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح یک و C = ۱۰.

با توجه به هدف مقاله که مقایسه عملکرد وب‌سرورها در سطوح مختلف منابع و کانتینرهاست، این دو معیار بهترین انعکاس را از تفاوت‌ها و گلوگاه‌های عملکردی فراهم می‌کنند.

٤- نتایج پیاده‌سازی

هر یک از نمودارهای ارائه‌شده شامل دو بخش است: نمودار الف (بالا) عملکرد وب‌سرور Nginx و نمودار ب (پایین) عملکرد وب‌سرور Apache را نشان می‌دهد. در تمامی این نمودارها، مقیاس تعداد درخواست‌ها 10^7 در نظر گرفته شده تا مقایسه عملکرد در سطوح مختلف منابع به‌وضوح قابل مشاهده باشد.

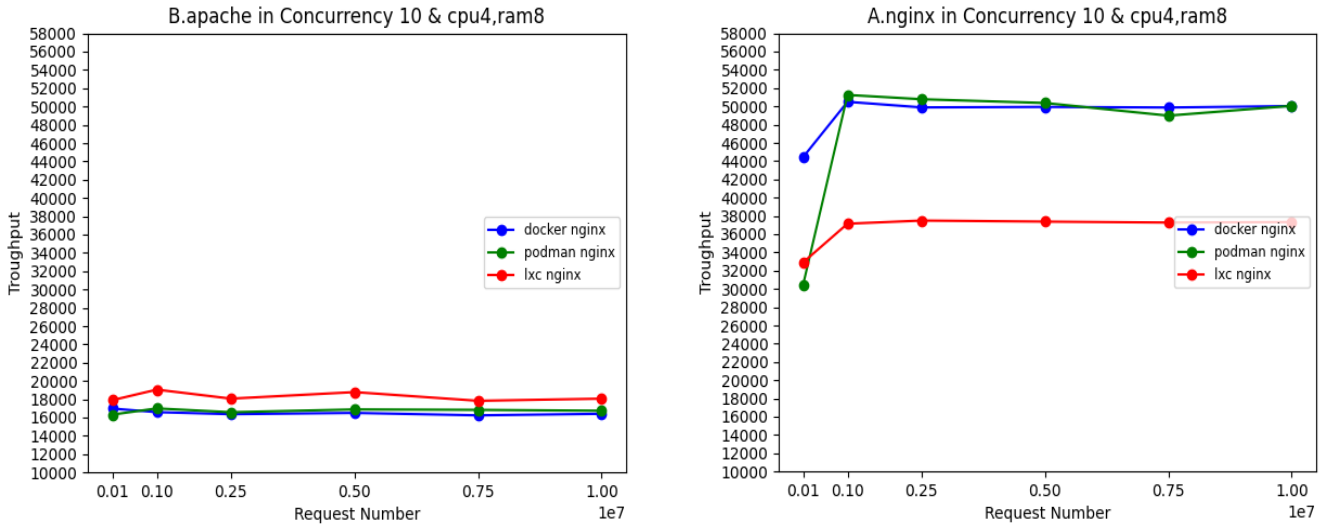
در سناریوی نمایش داده شده در شکل ۹، LXC در هر دو وب‌سرور بیشترین میزان گذردهی را نشان می‌دهد. دلیل این برتری آن است که LXC با داشتن پشته‌ای مستقیم و بدون واسطه به کرنل میزبان، می‌تواند پردازه‌ها و منابع خود را مستقیماً از طریق فراخوانی‌های سیستمی مدیریت کند. اما در پادمن و داکر، دایمن به‌عنوان یک پروکسی عمل می‌کند و تمامی فراخوانی‌های سیستمی کانتینر را دریافت و سپس به کرنل میزبان ارسال می‌کند. این روش، به‌ویژه در زمانی که منابع سخت‌افزاری محدودی در اختیار کانتینر است، سرعت مدیریت پردازه‌ها را کاهش می‌دهد. در پادمن و داکر، محدودیت‌های منابع توسط دایمن کانتینر مدیریت می‌شود، در حالی که در LXC این عمل از طریق مکانیزم‌های بومی لینوکس مانند محدودسازی در سطح سیستم‌عامل انجام می‌شود. این دسترسی مستقیم LXC به کرنل، به آن امکان می‌دهد که در نرم‌افزارهایی با معماری مبتنی بر تعداد بالای فراخوانی‌های سیستمی، عملکرد بهتری داشته باشد، زیرا فرایندهایی مانند باز و بسته کردن سوکت‌ها و مدیریت ارتباطات مستقیماً به کرنل ارجاع می‌شوند. این عملیات در کانتینرهایی که از دایمن برای پروکسی‌کردن درخواست‌ها و اعمال محدودیت‌ها استفاده می‌کنند، به دلیل واسطه‌گری دایمن کندتر انجام می‌گیرد. معماری Apache به شکلی طراحی شده که درخواست‌ها را با تخصیص به پردازه‌ها و نخ‌ها مدیریت کند؛ به این معنا که با افزایش درخواست‌ها، نیاز به فراخوانی‌های سیستمی بیشتری برای مدیریت پردازه‌ها و منابع دارد. در مقابل، معماری Nginx بر اساس تعداد مشخصی از پردازه‌های کارگر[1] و استفاده از مدل Event-Loop طراحی شده است. این رویکرد در نتایج نشان می‌دهد که در شرایط یکسان، Nginx معمولاً عملکرد بهتری نسبت به Apache دارد و فاصله قابل توجهی میان این دو وب‌سرور در بسیاری از سناریوهای مقایسه‌ای ایجاد نموده است.

با توجه به ارزیابی‌های انجام‌شده در میزان همروندی، به علت همروندی پایین و منابع زیاد، دایمن داکر و پادمن نیازی به مدیریت درخواست‌ها نمی‌بینند. همچنین معماری مبتنی بر Event-Loop وب‌سرور Nginx که وابستگی کمی به فراخوانی سیستمی دارد، توانسته در منابع بیشتر، نتایج بهتری را ارائه نماید.

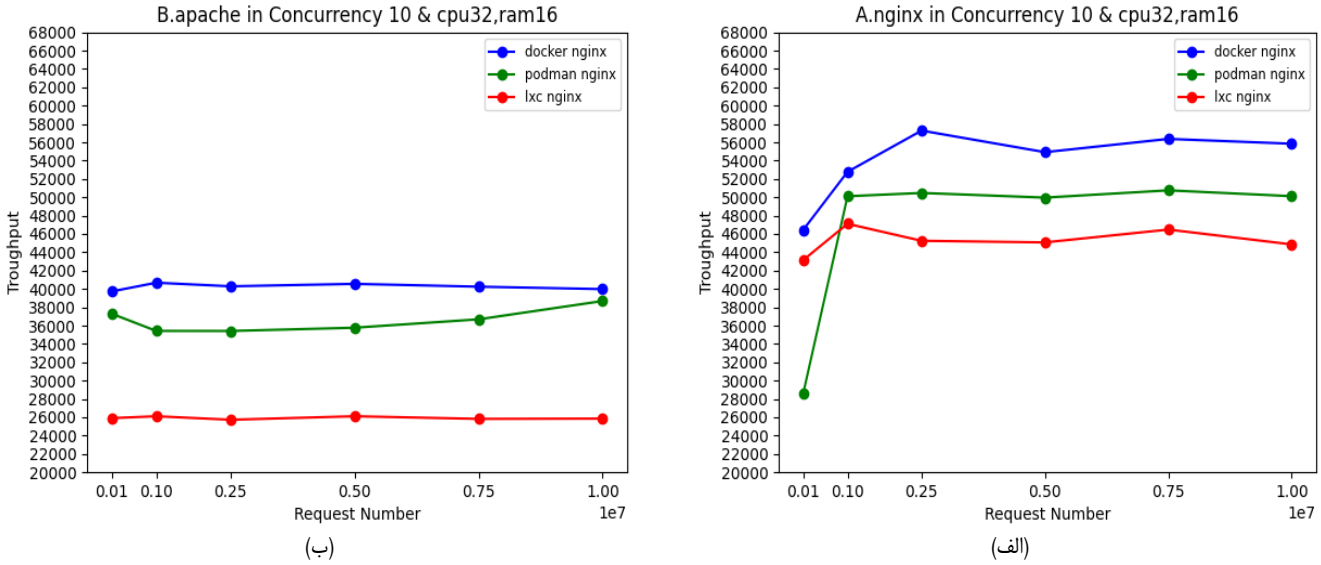
با توجه به شکل‌های ۱۰ و ۱۱، نتایج حاصل از آزمایش‌ها در سطح همروندی C = ۱۰ نشان می‌دهد که فناوری LXC در محیط‌هایی با منابع محدود، عملکرد بهتری نسبت به داکر و پادمن ارائه می‌دهد. این برتری عمدتاً به دلیل بهره‌گیری مستقیم LXC از قابلیت‌های بومی سیستم‌عامل لینوکس برای مدیریت منابع پردازه‌هاست. با افزایش منابع سخت‌افزاری در دسترس، داکر توانسته است از LXC پیشی بگیرد. دلیل این موضوع آن است که در معماری داکر، مدیریت فراخوانی‌های سیستمی به صورت غیرمستقیم و از طریق دایمن مستقر در سطح میزبان انجام می‌شود، نه در درون کانتینر که این مسئله منجر به کاهش سربار اجرای فراخوانی‌ها می‌شود. از نتایج به‌دست‌آمده همچنین می‌توان استنباط کرد که معماری‌های مبتنی بر فراخوانی‌های سیستمی، نظیر مدل‌های Process/Thread (مانند Apache) در محیط‌هایی که دایمن مسئول واسطه‌گری این فراخوانی‌هاست، عملکرد ضعیف‌تری دارند. در مقابل، معماری‌هایی که وابستگی کمتری به فراخوانی‌های سیستمی دارند، مانند Event-Loop (مورد استفاده در Nginx) در بستر فناوری‌هایی نظیر داکر و پادمن عملکرد بهتری از خود نشان می‌دهند.

به بیان دیگر، معماری‌های پردازه‌محور یا نخ‌محور به دلیل تولید تعداد زیادی از فراخوانی‌های سیستمی، در فناوری‌هایی مانند LXC که این فراخوانی‌ها مستقیماً در سطح کرنل اجرا می‌شوند، عملکرد مطلوب‌تری دارند. در مقابل، معماری‌های رویدادمحور[2] که تعداد کمتری فراخوانی سیستمی تولید می‌کنند، در بستر کانتینرهای مبتنی بر دایمن، مانند داکر و پادمن، عملکرد بهینه‌تری ارائه می‌دهند. نهایتاً باید توجه داشت که پادمن نیز به‌دلیل شباهت معماری با داکر، نتایجی مشابه ارائه می‌دهد؛ هرچند با تمرکز بیشتر بر امنیت ساختار دایمن در پادمن همانند داکر از فضای نام‌ها برای جداسازی کانتینرها استفاده می‌کند، اما تفاوت کلیدی در آن است که دایمن پادمن در فضای کاربر اجرا می‌شود، در حالی که دایمن داکر در

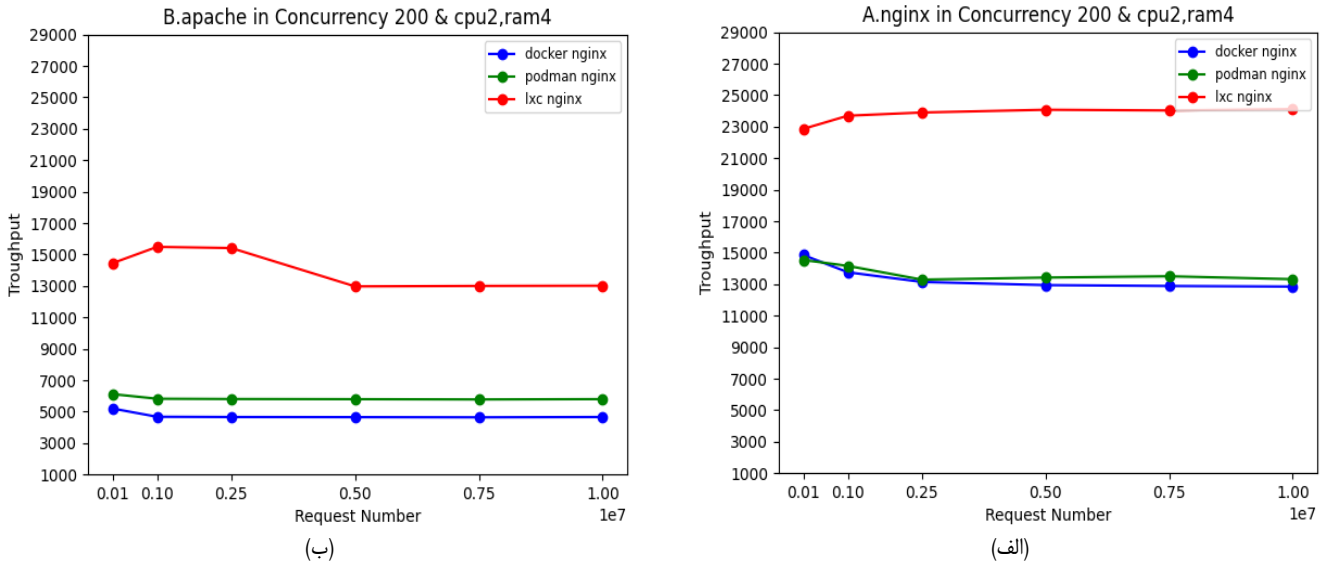
1. Worker Processes
2. Event-Based



شکل ۱۰: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح دو و C = ۱۰.



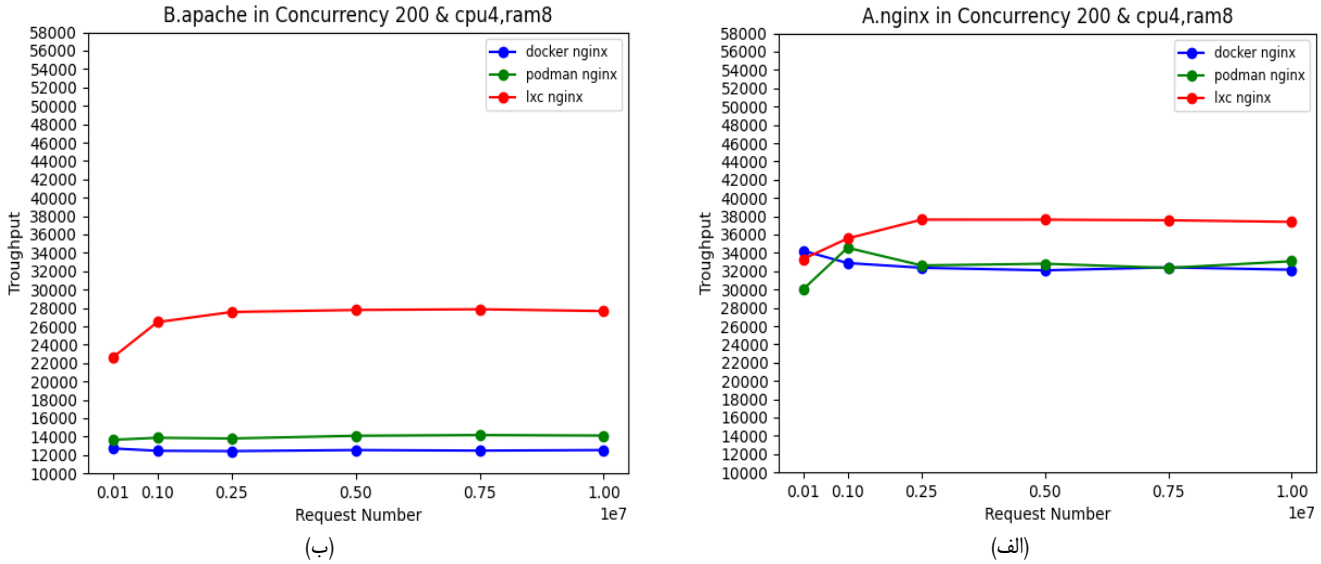
شکل ۱۱: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح سه و C = ۱۰.



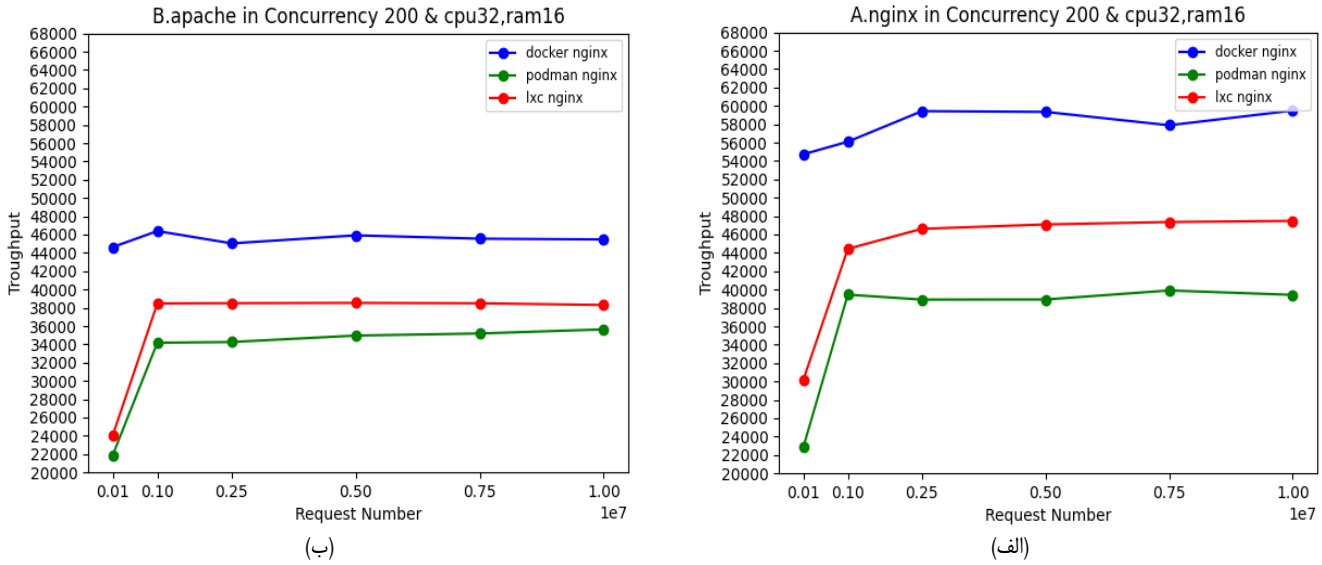
شکل ۱۲: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح یک و C = ۲۰۰.

فضای هسته قرار دارد. با این حال، هر دو در نحوه پروکسی‌کردن فراخوانی‌های سیستمی رفتار مشابهی دارند.

شکل‌های ۱۲ تا ۱۴ عملکرد وب‌سرورهای Apache و Nginx را بر بستر کانتینرهای داکر، پادمن و LXC در سه سطح منابع (کم، متوسط و



شکل ۱۳: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح دو و $C = ۲۰۰$.



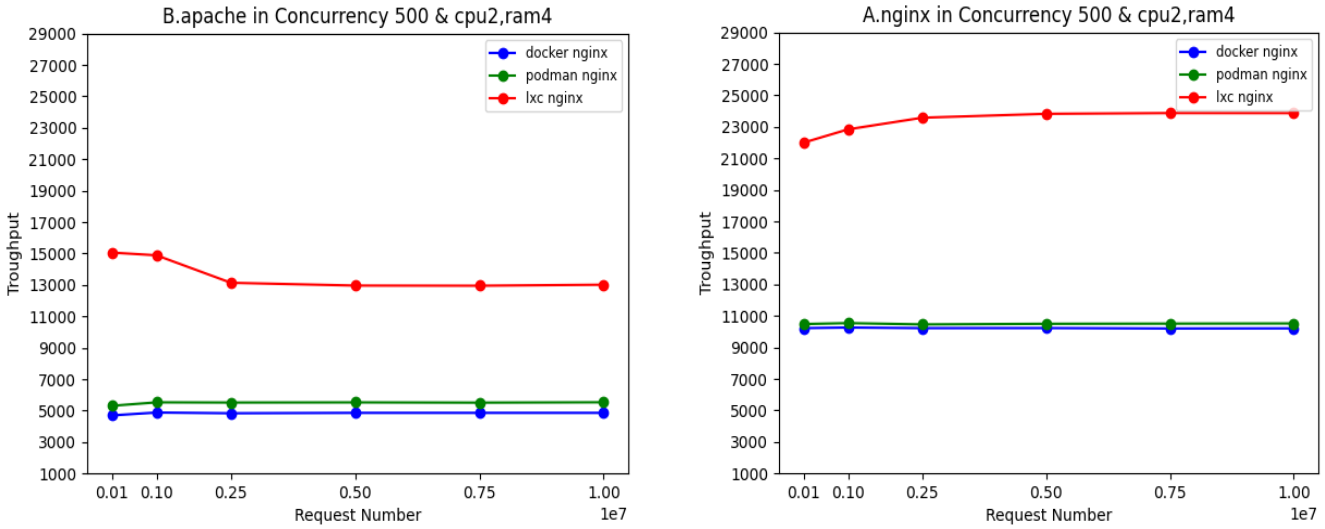
شکل ۱۴: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح سه و $C = ۲۰۰$.

کامل) و در شرایط همروندی $C = ۲۰۰$ نمایش می‌دهند. در شکل ۱۲، نتایج به‌دست‌آمده با نتایج همروندی $C = ۱۰$ مطابقت دارد و همان الگوهای عملکردی پیشین نیز در این سطح از همروندی مشاهده می‌شود. در شکل ۱۳، پیاده‌سازی Nginx بر روی LXC بهترین عملکرد را در میان سایر فناوری‌ها ثبت کرده است. افزایش تعداد ارتباط‌های هم‌زمان باعث افزایش تعداد فراخوانی‌های سیستمی در Nginx شده است. این موضوع نه به دلیل طراحی داخلی Nginx، بلکه ناشی از ماهیت عملکرد آن در مدیریت ارتباطات از طریق مکانیزم‌های سوکت است. به عبارت دیگر، وابستگی بالای پاسخ‌دهی Nginx به عملیات روی سوکت‌ها، موجب افزایش فراخوانی‌های سیستمی شده و نقش ساختار LXC در تسهیل این فراخوانی‌ها پررنگ‌تر می‌شود. در شکل ۱۴ نکته‌ای قابل توجه ظاهر می‌شود: تفاوت محسوس میان عملکرد داکر و پادمن در اجرای Nginx. این فاصله عملکردی، احتمالاً ناشی از تفاوت در پیاده‌سازی دایمن‌ها و نحوه مدیریت منابع در این دو فناوری است، به‌ویژه با توجه به آنکه پادمن برخلاف داکر از دایمن مستقل در فضای کاربر بهره می‌برد.

شکل‌های ۱۵ تا ۱۷ عملکرد وب‌سرورهای Apache و Nginx را بر روی کانتینرهای داکر، پادمن و LXC در سه سطح منابع و با همروندی $C = ۵۰۰$ نمایش می‌دهند. در شکل ۱۵، الگوی عملکرد مشاهده‌شده با نتایج آزمون‌های قبلی در سطوح همروندی $C = ۱۰$ و $C = ۲۰۰$ (با منابع سطح یک) مشابه است. با این حال با افزایش همروندی، فاصله نرخ گذردهی میان LXC و داکر به‌طور محسوسی افزایش یافته است. این موضوع نشان می‌دهد توانمندی LXC در استفاده مستقیم از ماژول‌های کرنل و انجام فراخوانی‌های سیستمی بدون واسطه در شرایط محدودیت منابع بسیار مؤثر و شایان توجه است.

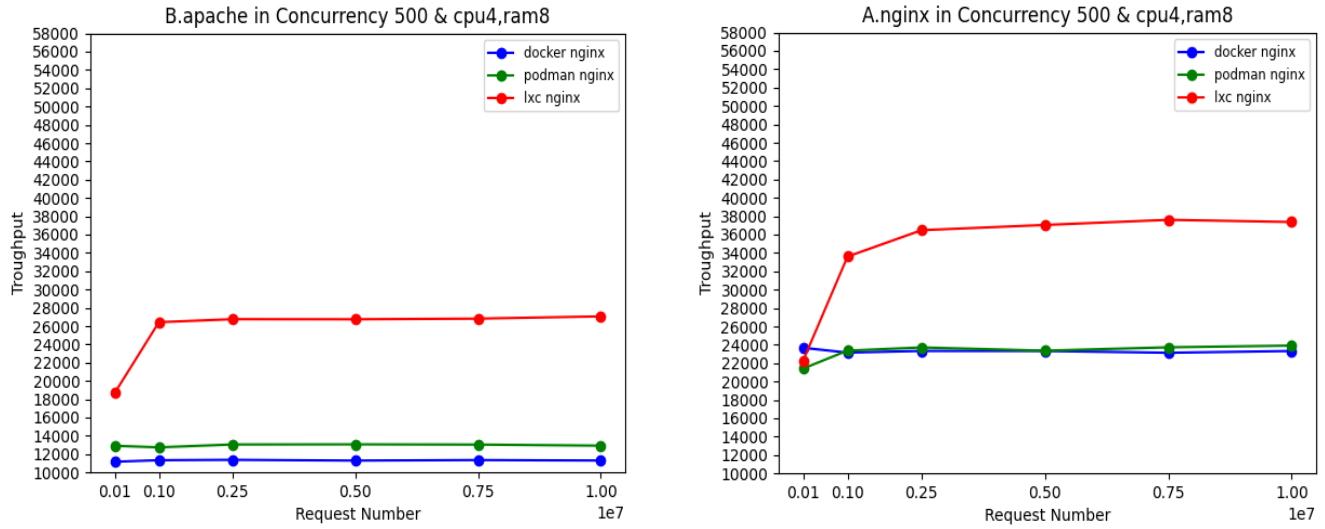
در شکل ۱۶ برخلاف آزمون‌های با همروندی پایین‌تر، اجرای Nginx بر بستر LXC بهترین عملکرد را ارائه کرده است. این نتیجه نشان‌دهنده کارایی بالای LXC در مدیریت فراخوانی‌های مرتبط با سوکت در شرایط بار بالا است. افزایش تعداد ارتباط‌های هم‌زمان نیازمند تعداد بیشتری فراخوانی سیستمی برای مدیریت اتصال‌هاست که LXC با حذف واسطه دایمن، مزیت محسوسی در این زمینه دارد. در نتیجه، اختلاف نرخ گذردهی بین LXC و دو فناوری دیگر، یعنی داکر و پادمن، در این سطح از همروندی بیشتر می‌شود.

در شکل ۱۷، رفتار کلی نتایج با آزمون‌های قبلی (در منابع سطح سه) سازگار است. با این حال، کاهش قابل توجهی در فاصله گذردهی بین LXC و داکر مشاهده می‌شود. این موضوع حاکی از آن است که با دسترسی به منابع سخت‌افزاری بیشتر، مزیت ذاتی LXC در حذف واسطه



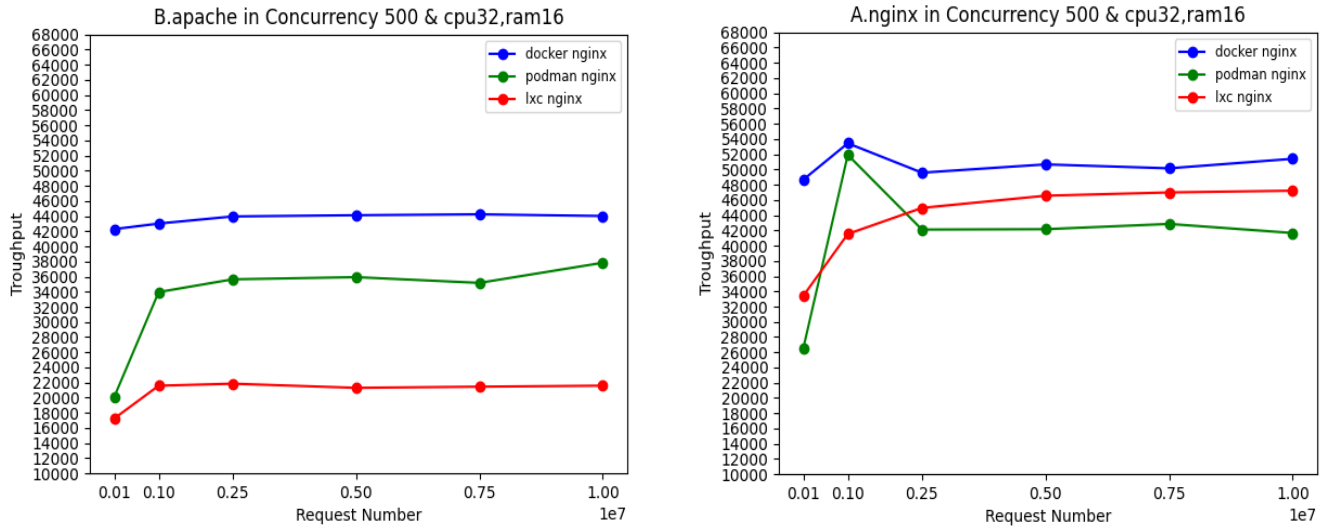
(الف) (ب)

شکل ۱۵: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح یک و $C = ۵۰۰$.



(الف) (ب)

شکل ۱۶: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح دو و $C = ۵۰۰$.

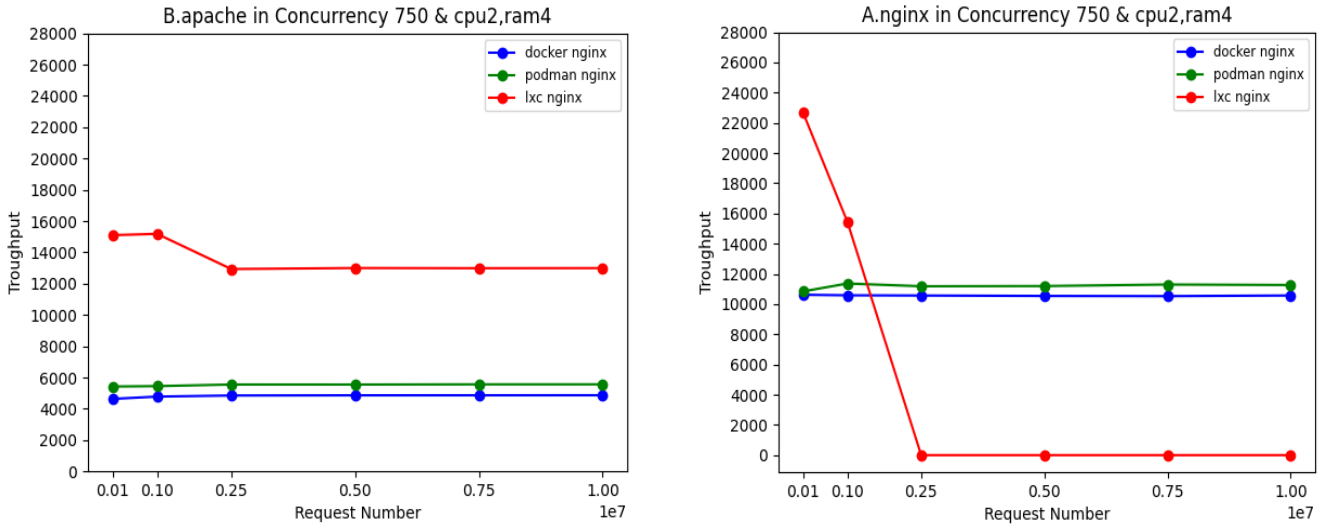


(الف) (ب)

شکل ۱۷: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح سه و $C = ۵۰۰$.

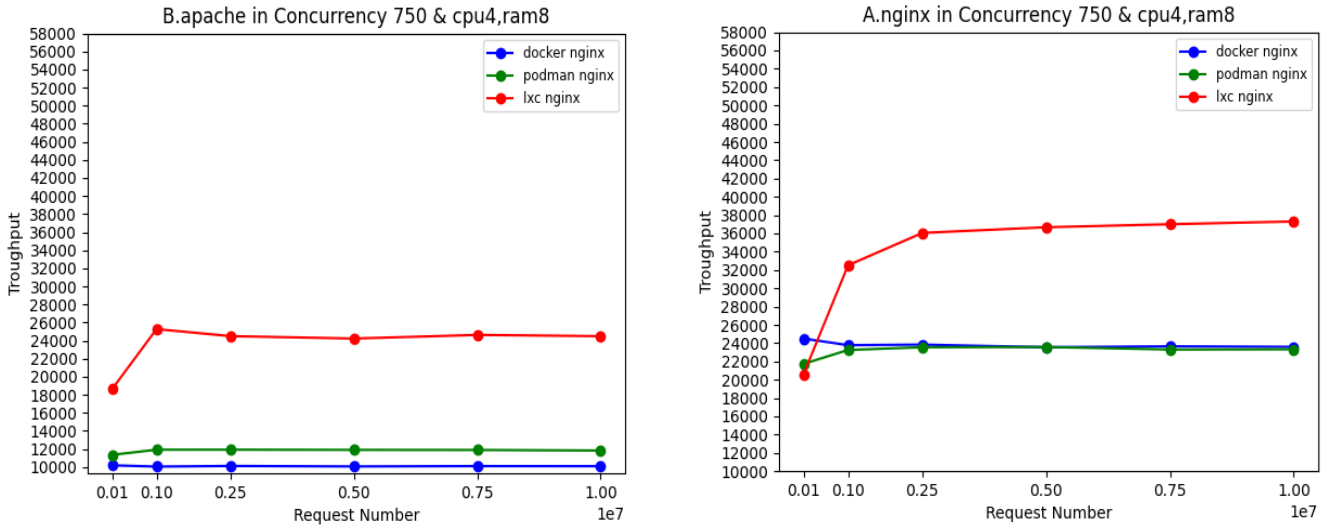
دایمن کاهش می‌یابد و فناوری‌های مبتنی بر دایمن مانند داکر می‌توانند بهینه‌تر از منابع استفاده کنند.

شکل‌های ۱۸ تا ۲۰ عملکرد وب‌سرورهای Apache و Nginx را بر روی کانتینرهای داکر، پادمن و LXC در سه سطح منابع و با همروندی



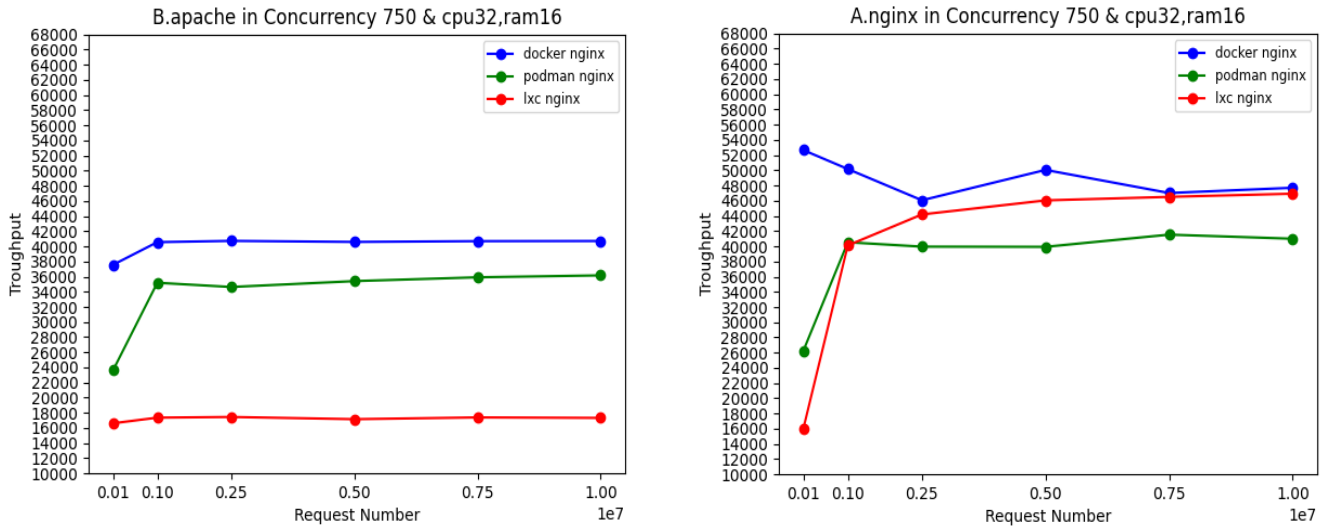
(الف) (ب)

شکل ۱۸: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح یک و $C = ۷۵۰$.



(الف) (ب)

شکل ۱۹: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح دو و $C = ۷۵۰$.



(الف) (ب)

شکل ۲۰: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح سه و $C = ۷۵۰$.

$C = ۷۵۰$ نشان می‌دهند. در شکل ۱۸، نتایج مربوط به Apache همانند مشاهدات قبلی در سطوح $C = ۱۰$ و $C = ۲۰۰$ است؛ به‌طوری که فناوری LXC بالاترین نرخ گذردهی را برای این وب‌سرور ثبت کرده است. با این حال در مورد Nginx، آزمون با شکست مواجه شده و به پایان نرسیده

```
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
```

شکل ۲۱: وضعیت ابتدایی ارتباط‌های TCP در LXC قبل از شروع آزمون.

است. در این حالت به دلیل بار کاری بالا بر روی کانتینر، وب‌سرور دچار کرش شده و پیام RST[1] به سمت ابزار Apache Benchmark ارسال شده است؛ این پیام نشان‌دهنده قطع ناگهانی اتصال از سمت سرور است. در واقع، Apache Benchmark منتظر دریافت پاسخ از Nginx باقی می‌ماند، اما به دلیل ناتوانی وب‌سرور در مدیریت حجم بالای درخواست‌ها، هیچ پاسخی بازگردانده نمی‌شود و ارتباطها ریست می‌شوند. تحلیل نتایج نشان می‌دهد که در بار اولیه (حدود صدهزار درخواست)، Nginx عملکرد مناسبی داشته، اما با افزایش تعداد درخواست‌ها به بیش از دو میلیون، توانایی مدیریت پایدار ارتباطها را از دست داده است. از منظر فنی، این اختلال به محدودیت‌های پیکربندی کرنل در مدیریت ارتباطهای TCP بازمی‌گردد. هر اتصال TCP پس از پایان تبادل داده، برای مدتی در وضعیت Time-Wait باقی می‌ماند. با افزایش همروندی، تعداد ارتباطاتی که در این وضعیت باقی می‌مانند به‌تدریج افزایش یافته و به سقف تعیین‌شده در تنظیمات سیستم‌عامل می‌رسد. از آنجا که ارتباطها در وضعیت Establish ثابت باقی مانده‌اند، امکان ایجاد اتصال جدید وجود ندارد و سرور عملاً از پاسخ‌گویی بازمی‌ماند. اگرچه هدف این پژوهش بررسی عملکرد فناوری‌های کانتینری در لایه اپلیکیشن است، همان گونه که مشاهده شد، برخی نتایج آزمون‌ها مستقیماً تحت تأثیر محدودیت‌های کرنل و تنظیمات سیستم‌عامل قرار می‌گیرند. در بخش‌های بعدی مقاله، این موضوع به‌طور مستند و تحلیلی مورد بررسی قرار خواهد گرفت.

شکل ۲۱ وضعیت ابتدایی ارتباطهای TCP در کانتینر LXC را پیش از آغاز آزمون و در زمان اجرای وب‌سرور Nginx با منابع سطح یک نمایش می‌دهد. در این وضعیت پایه، پیش از شروع بارگذاری، تنها یک اتصال از نوع Foreign، یک اتصال در وضعیت Established (که نشان‌دهنده ارتباط فعال برای دسترسی به داخل کانتینر است) و پنج اتصال در وضعیت Listen ثبت شده‌اند. این وضعیت نمایانگر شرایط پایدار شبکه پیش از اعمال بار کاری و آغاز آزمون همروندی است.

پس از آغاز آزمون با همروندی $C = ۷۵۰$، طبق آنچه در شکل ۲۲ آمده است، تعداد ارتباطهای فعال به‌درستی به عدد ۷۵۰ می‌رسد. با این حال، به‌مرور شاهد افزایش تعداد اتصال‌هایی هستیم که در وضعیت Time-Wait باقی می‌مانند. این وضعیت به دلیل ماهیت پروتکل TCP و مکانیزم طراحی‌شده برای مدیریت صحیح بسته‌ها پس از پایان یک ارتباط به وجود می‌آید. حالت Time-Wait برای جلوگیری از بروز تداخل در ارتباطهای جدید و اطمینان از پاک‌سازی کامل سوکت‌ها، اتصال را برای مدتی پس از بسته‌شدن نگه می‌دارد. این طراحی که با هدف کاهش سربار ناشی از فرایند three-way handshake و تسهیل استفاده مجدد از سوکت‌ها انجام شده، در شرایط بار بالا ممکن است که منجر به اشباع منابع شود. در شکل ۲۲ قابل مشاهده است که با افزایش تدریجی تعداد ارتباطهای در وضعیت Time-Wait و محدودیت ظرفیت منابع سیستم‌عامل، توانایی پذیرش درخواست‌های جدید به‌طور چشمگیری کاهش یافته و سرور عملاً از پاسخ‌گویی بازمی‌ماند. نهایتاً همان طور که در شکل ۲۳ نشان داده شده است، ابزار Apache Benchmark برای ادامه آزمون ناچار به بازنشانی (reset) سوکت‌ها می‌شود، اما به دلیل

1. Reset

```
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
    750 ESTABLISHED
   1021 TIME_WAIT
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
    757 ESTABLISHED
   1239 TIME_WAIT
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
     31 SYN_RECV
    715 ESTABLISHED
   2108 TIME_WAIT
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
     28 SYN_RECV
    716 ESTABLISHED
   2503 TIME_WAIT
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
     20 SYN_RECV
    713 ESTABLISHED
  13264 TIME_WAIT
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
     23 SYN_RECV
    716 ESTABLISHED
  13545 TIME_WAIT
root@ubuntu-nginx:~# netstat -nat | awk '{print $6}' | sort | uniq -c | sort -n
      1 Foreign
      1 established)
      5 LISTEN
    716 ESTABLISHED
  14173 TIME_WAIT
```

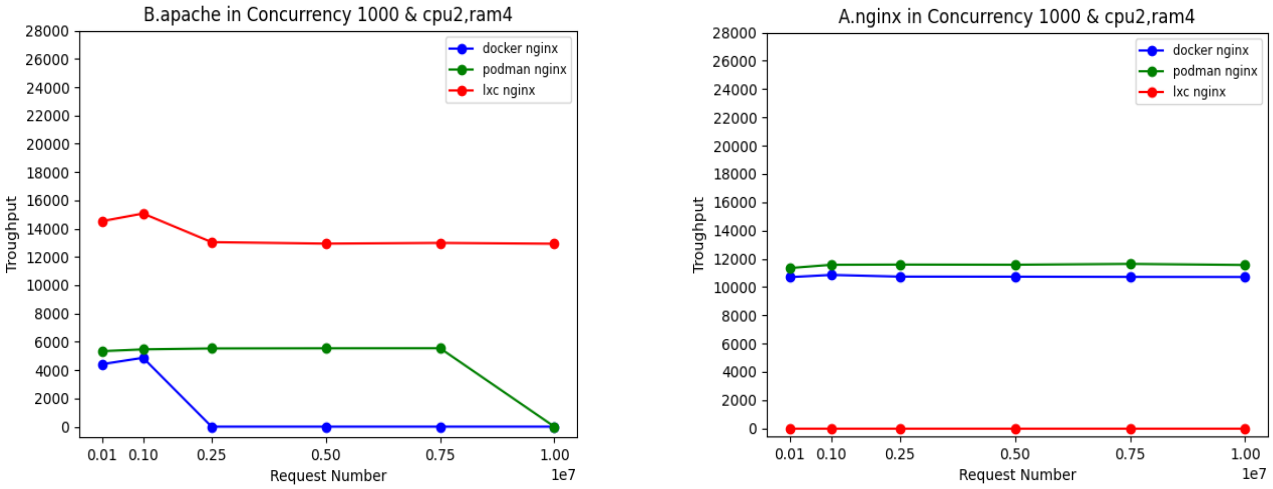
شکل ۲۲: بررسی وضعیت حالت‌های ارتباط‌های TCP در هنگام انجام آزمون.

ناکامی در بازکردن ارتباط‌های جدید، اجرای آزمون ناتمام باقی می‌ماند.

با شکست این آزمون، سؤالی که مطرح می‌شود این است که مشکل از معماری LXC مبتنی بر فراخوانی سیستمی یا مدل Event-Loop در وب‌سرور Nginx است؟ پاسخ منفی است! هر دوی این معماری‌ها برای وب‌سرورها کاملاً مناسب هستند. معماری LXC که محدودیت‌های سخت‌افزاری را با ویژگی‌های بومی لینوکس اعمال می‌کند، در مقایسه با مدیریت منابع توسط دایمن در داکر و پادمن، سربار کمتری دارد و روش بهینه‌تری است. البته روش مبتنی بر Event-Loop وب‌سرور Nginx نیز کمترین فشار را از نظر تعداد فراخوانی سیستمی اعمال می‌کند و صرفاً برای مدیریت ارتباطها از فراخوانی سیستمی استفاده می‌کند. ولی شکست آزمون بالا ارتباطی با کانتینر ندارد. کانتینر به نوعی از مجازی‌سازی گفته می‌شود که در آن تمامی لایه Kernel space بین تمامی کانتینرها با هدف کم‌کردن سربارهای هر یک از ماشین‌های مجازی به اشتراک گذاشته شده است. در حالت پیش‌فرض با توجه به پارامترهای کرنل، تمامی ارتباطها ملزم به پایان‌دادن خود پس از انتظار در حالت Time-Wait هستند و به صورت پیش‌فرض در کرنل امکان استفاده دوباره از یک ارتباط TCP به عنوان سرور غیرفعال است. با توجه به همپوشانی مناسب LXC و وب‌سرور Nginx و نرخ گذردهی بسیار بالا، افزایش نرخ همروندی موجب سرعت در پاسخگویی به درخواست‌ها و افزایش سریع تعداد ارتباطهای وضعیت Time-Wait تا پایان‌یافتن زمان انتظار بسته‌شدن ارتباط شده است. این افزایش تعداد ارتباط در نهایت به سقف تعداد ارتباطها می‌رسد و نهایتاً ارتباط قطع می‌شود و آزمون کامل نمی‌شود. این امر برخلاف انتظار نشان از سرعت و بهره‌وری بالای این پشته معماری وب‌سروری/کانتینری است که در منابع بالا و همروندی بالا، نتایج خوب همین پشته معماری LXC+Nginx مشهود است. ولی با توجه به ایزوله‌سازی‌ها در لایه سیستم‌عامل، تکنولوژی LXC برخلاف داکر دیگر توانایی استفاده از تمامی منابع و دسترسی‌های BareMetal را دارا نیست و این امر موجب شکست در این آزمون می‌شود؛ در حالی که در حالت استفاده بدون محدودیت از منابع (منابع

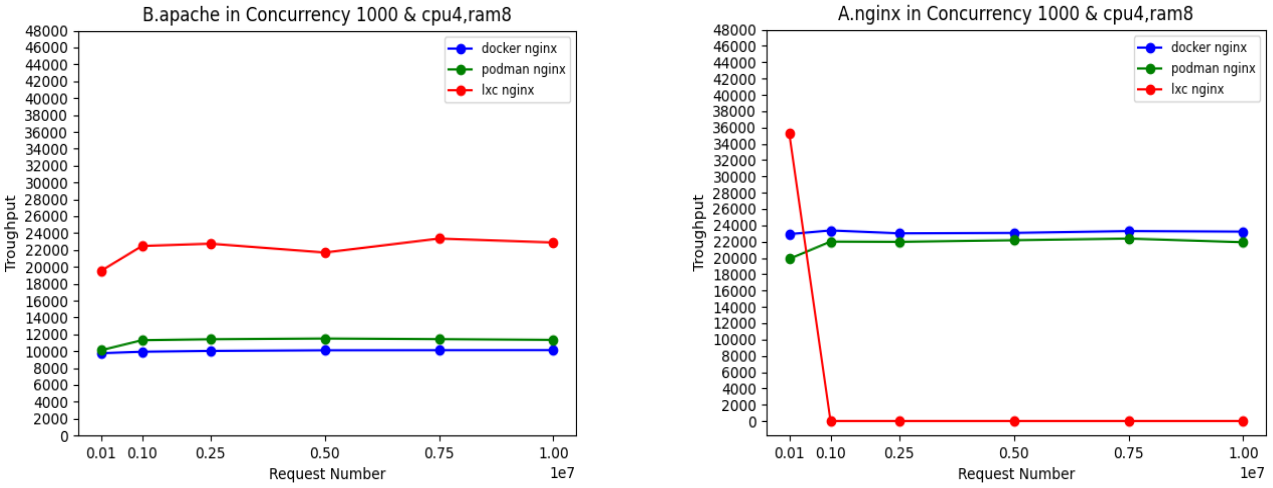
```
umz@umz-uni:~/proj/scirpts/result$ ab -k -n 10000000 -c 750 "http://10.211.216.145:80/"
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 10.211.216.145 (be patient)
Completed 1000000 requests
Completed 2000000 requests
apr_socket_recv: Connection reset by peer (104)
Total of 2685854 requests completed
```

شکل ۲۳: شکست آزمون.



(الف) (ب)

شکل ۲۴: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح یک و $C = ۱۰۰۰$.

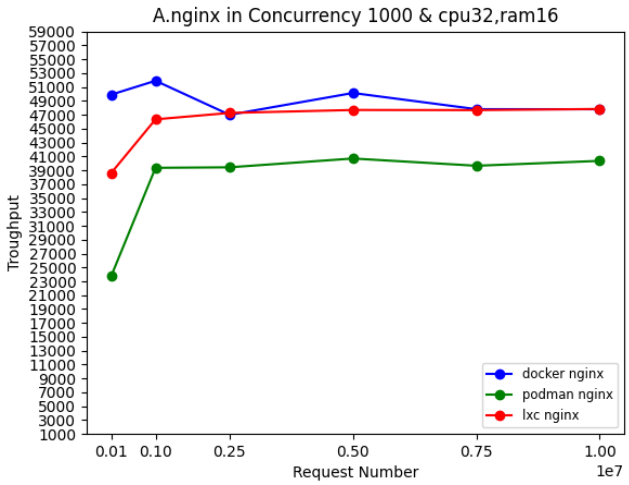


(الف) (ب)

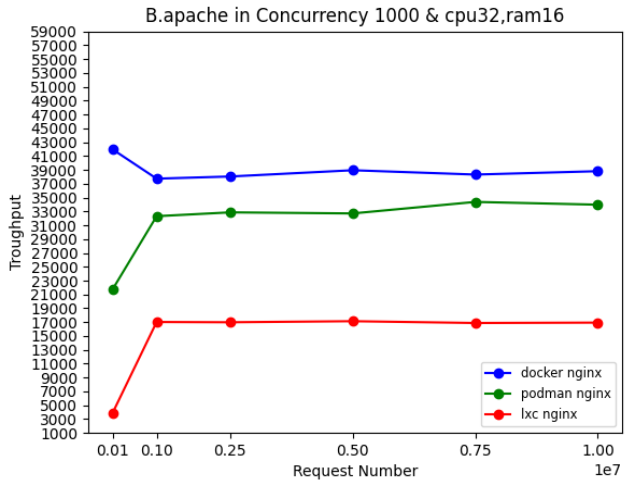
شکل ۲۵: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح دو و $C = ۱۰۰۰$.

سطح ۳) عملکرد مناسب این پشته و تکنولوژی LXC دیده می‌شود. شاید اگر این نقص یا به عبارتی مزیت LXC مرتفع شود، به‌سرعت جایگاه مناسبی در معماری‌های مبتنی بر CNCF[1] پیدا کند. به طور خلاصه ایزوله‌سازی در لایه داخلی خود کانتینر موجب ناتمام‌شدن این آزمون شده است؛ در حالی که همین معماری به علت سرعت بالای پاسخ به فراخوانی‌ها تا قبل از این شرایط، نتایج خوبی را در همین تعداد منابع از خود نشان داده بود. در واقع، عدم هماهنگی بین سرعت بالای پاسخ‌دهی وب‌سرور در لایه اپلیکیشن و سرعت پایین‌تر بسته‌شدن ارتباطهای TCP منجر به افزایش تعداد ارتباطهای باز در وضعیت Time-Wait می‌شود. از آنجا که همزمانی به حفظ ۷۵۰ ارتباط برقرار الزام دارد، این ارتباطها باید سریع پاسخ دریافت کنند و بسته شوند، اما سرعت پایین‌تر در بستن این ارتباطها باعث تجمع آنها در حالت Time-Wait می‌شود. نتیجه این وضعیت، کرش‌کردن وب‌سرور است. در لینوکس می‌توان با تغییردادن تنظیمات کرنل TCP مانند کاهش زمان Time-Wait و سایر پارامترهای مرتبط، این مشکل را رفع کرد. با تنظیم پارامترهایی مانند net.ipv4.TCP_tw_recycle، net.ipv4.TCP_fin_timeout، net.ipv4.TCP_tw_reuse، net.core.somaxconn و دیگر پارامترهای مشابه، می‌توان این چالش را به طور کامل حل کرد. با این حال به دلیل استفاده از شرایط یکسان در آزمون و اهمیت تنظیم این بخش‌ها توسط خود فناوری‌های کانتینری، این تنظیمات در اینجا مورد استفاده قرار نگرفتند. شکل‌های ۲۴ تا ۲۶ عملکرد وب‌سرورها بر روی کانتینرها با منابع مختلف در همروندی $C = ۱۰۰۰$ را نشان می‌دهند. شکل ۲۴

1. Cloud Native Computing Foundation



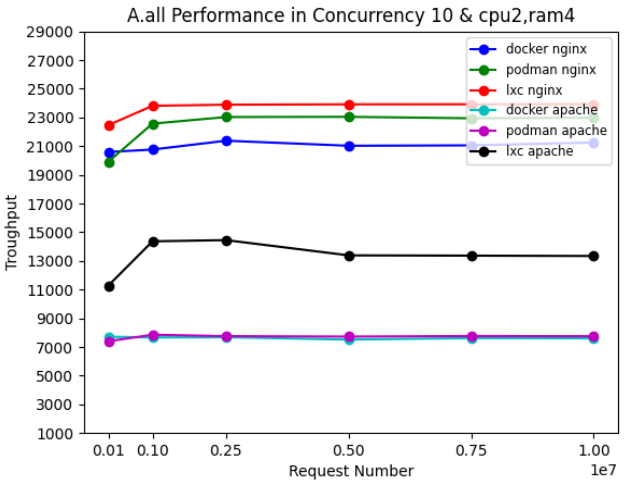
(الف)



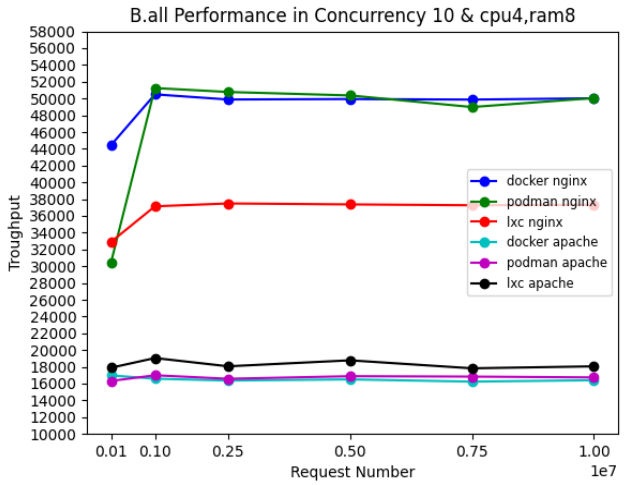
(ب)

شکل ۲۶: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع سطح سه و $C=۱۰۰۰$.

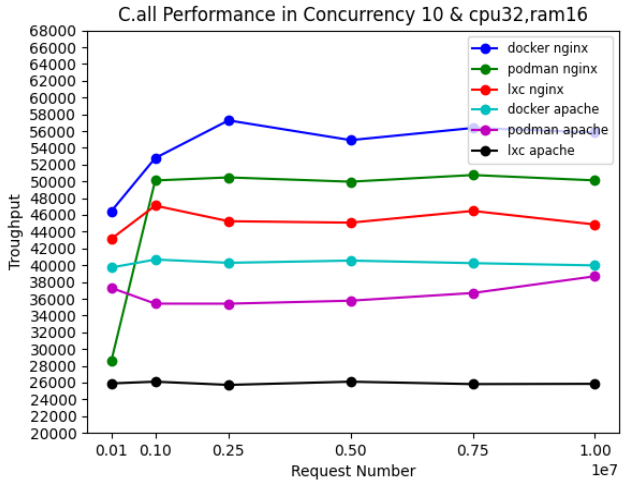
تصدیق نکات مطرح‌شده در آزمون قبلی برای همین سطح از منابع است. عملکرد بهتر LXC برای Apache و شکست‌خوردن آزمون در مورد Nginx همچنان برقرار است. همچنان داکر و پادمن تمامی آزمون‌ها در پیاده‌سازی Nginx را با موفقیت پشت سر گذاشته‌اند؛ اما در مورد Apache، هر دو نتوانستند تمامی آزمون‌ها را به‌درستی پشت سر بگذارند. در شکل ۲۵ مشاهده می‌شود که LXC عملکرد بهتری برای Apache دارد، ولی در مورد Nginx با وجود افزایش منابع سخت‌افزاری قادر به اتمام آزمون نیست. همچنان داکر و پادمن تمامی آزمون‌های مرتبط با Nginx را پاس کرده‌اند. در شکل ۲۶ با توجه به افزایش منابع و عدم نیاز به مدیریت فراخوانی‌های سیستمی توسط دایمن خود، داکر بالاترین نرخ گذردهی را ثبت نموده است. البته که در مورد Nginx عملکرد مشابه داکر و LXC در ارائه بیشترین میزان نرخ گذردهی مشاهده می‌شود. مشخص است با برداشته‌شدن محدودیت‌های منابع، LXC توانسته از تمامی منابع سخت‌افزاری در داخل کانتینر استفاده کند و برخلاف آزمون‌هایی با منابع کمتر، کرش نمی‌کند و مطابق انتظار نتایج بهتری حاصل می‌شود. مطابق توضیحات بخش ایزوله‌سازی کانتینر، LXC مانند پادمن و داکر (در همه حالات منابعی) از داخل کانتینر می‌تواند تمام منابع سیستمی را ببیند که این امر موجب پشت سر گذاشتن این آزمون در این سطح از منابع می‌شود. شکل‌های ۲۷ تا ۳۰ به مقایسه عملکرد دو وب‌سرور Apache و Nginx در بستر سه فناوری کانتینرسازی داکر، پادمن و LXC می‌پردازند. در هر



(الف)



(ب)

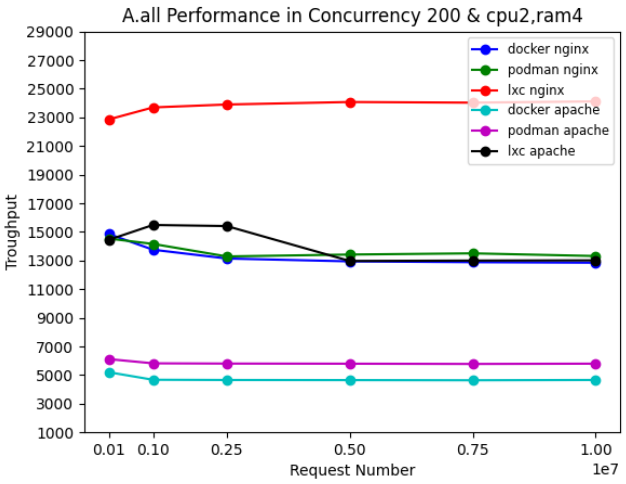


(ج)

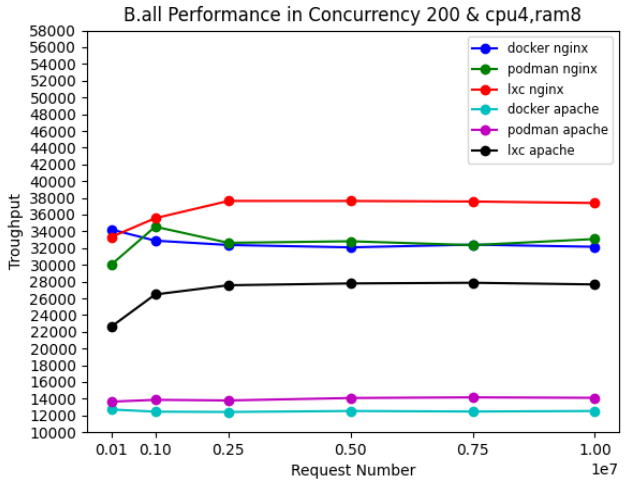
شکل ۲۷: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع مختلف با $C=۱۰$.

نمودار، یکی از سطوح همروندی ($C=۱۰$ تا $C=۱۰۰۰$) بررسی شده است. بررسی‌ها نشان می‌دهند که وب‌سرور Apache، علی‌رغم سازگاری معماری آن با LXC از حیث بهره‌گیری از فراخوانی‌های سیستمی، نتوانسته در هیچ یک از سطوح همروندی، نرخ گذردهی قابل توجهی را بر بستر LXC ثبت کند. این موضوع احتمالاً به نحوه مدیریت داخلی پردازه‌ها در Apache و محدودیت‌های اعمال‌شده در لایه ایزوله‌سازی LXC بازمی‌گردد. در مقابل، وب‌سرور Nginx با

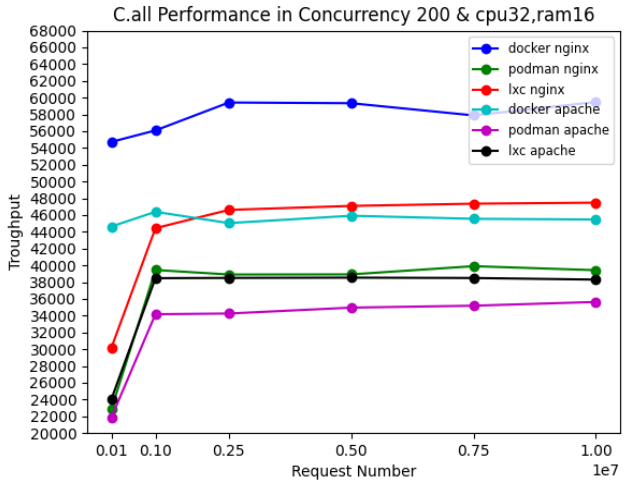
معماری



(الف)



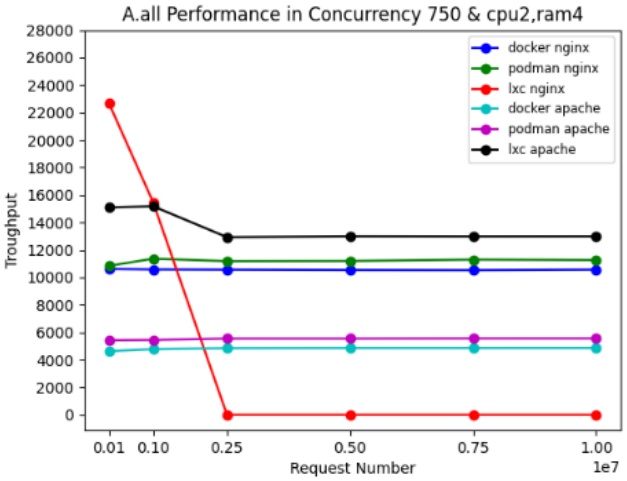
(ب)



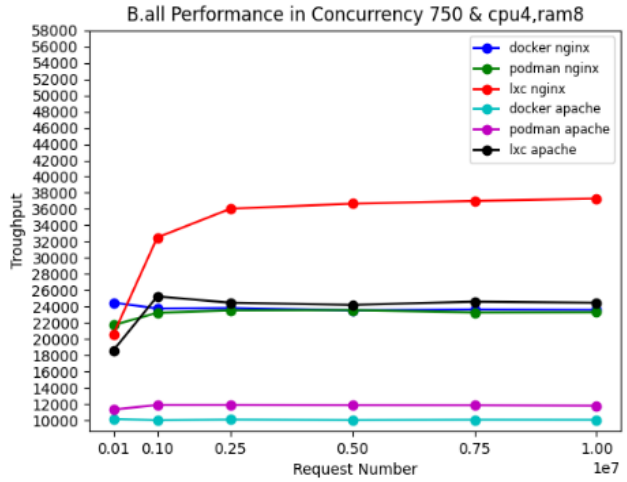
(ج)

شکل ۲۸: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع مختلف با $C = ۲۰۰$.

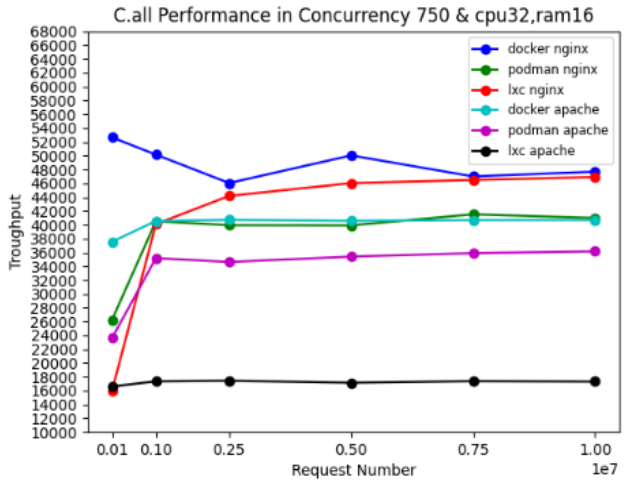
مبتنی بر رویداد، در سطوح پایین‌تر همروندی، عملکرد بسیار بهتری در تمامی فناوری‌های کانتینرسازی از خود نشان داده است. بهره‌گیری از Event-Loop در این وب‌سرور باعث کاهش تعداد فراخوانی‌های سیستمی، استفاده بهینه‌تر از منابع و در نتیجه افزایش نرخ پاسخ‌گویی شده است. این نتایج بار دیگر مؤید آن است که انتخاب وب‌سرور مناسب در کنار فناوری کانتینرسازی سازگار با معماری آن، نقشی کلیدی در عملکرد کلی سیستم دارد. به‌ویژه در سطوح پایین همروندی، تأثیر معماری داخلی اپلیکیشن بر کارایی نهایی بیشتر از نوع فناوری کانتینری مشهود است.



(الف)



(ب)



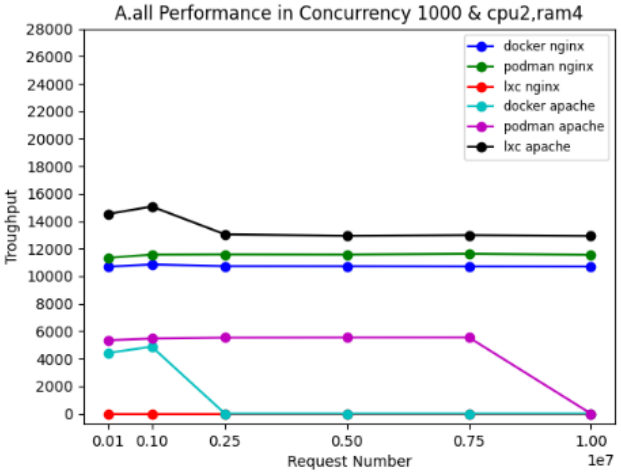
(ج)

شکل ۲۹: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع مختلف با $C = ۷۵۰$.

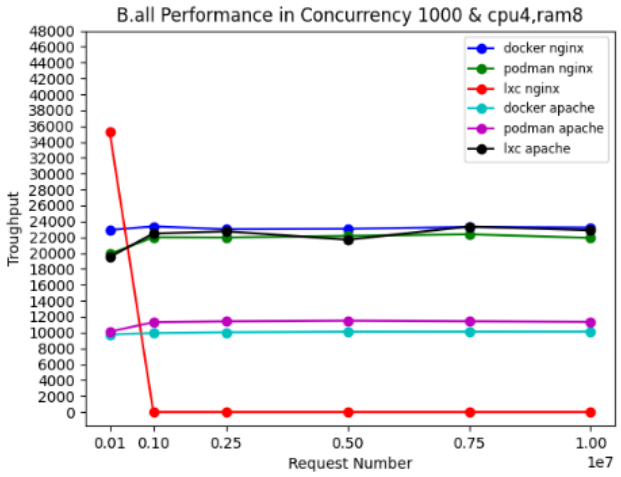
۵- نتیجه‌گیری

در مجموع، در محیط‌های با منابع محدود، کانتینر LXC عملکرد بهتری نسبت به داکر و پادمن دارد. با این حال در آزمایش‌هایی با همروندی بیش

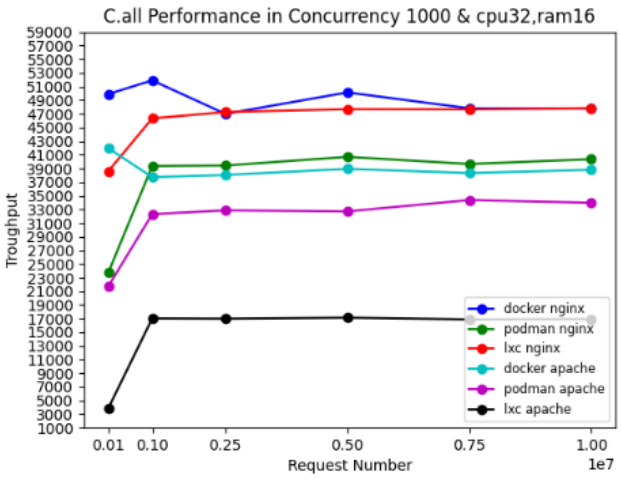
از ۵۰۰، LXC نتوانسته است تمامی آزمون‌ها را به طور کامل پشت سر بگذارد. استفاده از ویژگی‌های بومی لینوکس در لایه سیستم‌عامل، علاوه



(الف)



(ب)



(ج)

شکل ۳۰: بررسی عملکرد وب‌سرورهای Apache و Nginx بر روی کانتینرها با منابع مختلف با C = ۱۰۰۰.

بر کاهش سربار پردازشی به دلیل ایزوله‌سازی داخلی کانتینر بدون واسطه دایمن، محدودیت‌هایی در تعداد سوکت‌ها ایجاد می‌کند. با افزایش سرعت پاسخ‌دهی به درخواست‌ها، تعداد ارتباطات در وضعیت Time-Wait افزایش می‌یابد که این مسئله می‌تواند منجر به کرش وب‌سرور در همروندی بالا شود. در این شرایط، داکر در همروندی بالا نسبت به LXC و پادمن عملکرد بهتری نشان داده است. هنگامی که حداقل منابع در دسترس باشد و سطح همروندی بالا باشد، وب‌سرور Apache روی LXC بیشترین نرخ گذردهی را دارد. همچنین با تنظیمات خاص کرنل می‌توان ترکیب Nginx و LXC را به یک پشته بهینه برای محیط‌های با منابع محدود تبدیل کرد.

با توجه به مدل شش‌لایه‌ای معرفی‌شده در بخش ۳، نتایج تجربی نشان دادند لایه Orchestration تعیین‌کننده مدیریت بار و مقیاس‌پذیری سیستم است، لایه Container Engine مسئول تعامل ایمن و کاربرمحور با سیستم است، لایه Runtime Interface نقش مهمی در تطبیق‌پذیری با انواع محیط‌های اجرایی ایفا می‌کند، Container Runtime بر چرخه حیات و کارایی اجرای کانتینر اثر مستقیم دارد، لایه STD واسطی مؤثر برای کاهش سربار بین بخش‌های سطح بالا و پایین است و نهایتاً Kernel Level عامل نهایی در تخصیص منابع و تعیین رفتار واقعی کانتینرها است. ترکیب این لایه‌ها، چارچوب جامعی برای تحلیل فنی تعامل بین وب‌سرورها و فناوری‌های کانتینری در سناریوهای مختلف ایجاد می‌کند.

در شرایط حداکثر همروندی و منابع فراوان، ترکیبات Docker-Nginx و LXC-Nginx بهترین عملکرد را نشان داده‌اند و می‌توانند با توجه به منابع در دسترس به‌عنوان مدل‌های پیاده‌سازی مناسب در نظر گرفته شوند. اگر منابع محدود باشند، Apache بر روی LXC گزینه مناسبی است و اگر منابع کافی در اختیار باشد، Nginx بر روی LXC یا داکر انتخابی ایده‌آل خواهد بود. بررسی‌ها نشان دادند که معماری مبتنی بر پردازه و نخ وب‌سرور Apache نتوانسته است نظریه C۱۰K را در سطح همروندی بالا پشت سر بگذارد و این آزمون در سطح C۱K (همروندی ۱۰۰۰) پایان یافته است. با این حال، Apache توانسته است در شرایطی با حداقل منابع و در برخی کانتینرها نرخ گذردهی بالایی را به ثبت برساند. پادمن نیز به‌طور کلی عملکرد متوسطی از خود نشان داده و نتایج حاکی از آن است که این فناوری با هدف افزایش امنیت نسبت به داکر طراحی شده و ایزوله‌سازی بهتری ارائه می‌دهد. نتایج این بررسی نشان می‌دهد که LXC در محیط‌های با منابع محدود، به دلیل ارتباط مستقیم با کرنل و استفاده بهینه از منابع، عملکرد بهتری نسبت به داکر و پادمن دارد. این ویژگی به‌ویژه در وب‌سرور Apache که مبتنی بر پردازش‌های همزمان است، بیشتر مشهود است. در مقابل با افزایش منابع سخت‌افزاری، داکر می‌تواند مدیریت مؤثرتری بر این منابع داشته باشد و به تنظیمات سیستمی پیچیده نیاز کمتری دارد. برای محیط‌هایی با تعداد بالای ارتباط‌های همزمان (مانند Nginx با معماری Event-Loop)، داکر و پادمن به دلیل ساختار مبتنی بر دایمن، نرخ گذردهی بالاتری را ارائه می‌دهند. این نتایج بیانگر آن است که انتخاب فناوری کانتینر مناسب به نوع وب‌سرور و مقدار منابع سخت‌افزاری در دسترس بستگی دارد. در محیط‌های با منابع محدود، LXC گزینه مناسبی است و در محیط‌های با منابع بیشتر، داکر انتخاب بهتری خواهد بود. به‌طور کلی، یافته‌های این مقاله به کاربران کمک می‌کند تا بر اساس نیازها و محدودیت‌های خود، بهترین فناوری مجازی‌سازی را برای وب‌سرورهای خود انتخاب کنند.

مراجع

[1] N. Fazuludeen, S. S. Banu, A. Gupta, and V. Swathi, "Challenges and issues of managing the virtualization environment through Vmware Vsphere," *Nanotechnology Perceptions*, vol. 20, no. S1, pp. 281-292, 2024.

[2] A. M. Potdar, D. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419-1428, 2020.

[3] S. Lozano, T. Lugo, and J. Carretero, "A comprehensive survey on the use of hypervisors in safety-critical systems," *IEEE Access*, vol. 11, pp. 36244-36263, 2023.
[4] A. Bhardwaj and C. R. Krishna, "Virtualization in cloud computing: moving from hypervisor to containerization-a survey," *Arabian J. for Science and Engineering*, vol. 46, pp. 8585-8601, 2021.
[5] R. Ranjan, I. S. Thakur, G. S. Aujla, N. Kumar, and A. Y. Zomaya, "Energy-efficient workflow scheduling using container-based virtualization in software-defined data centers," *IEEE Trans. on Industrial Informatics*, vol. 16, no. 12, pp. 7646-7657, Dec. 2020.
[6] K. Wang, *et al.*, "Characterizing and optimizing Kernel resource isolation for containers," *Future Generation Computer Systems*, vol. 141, pp. 218-229, Apr. 2023.
[7] G. Rodriguez, *et al.*, "Understanding and addressing the allocation of microservices into containers: a review," *IETE J. of Research*, vol. 70, no. 4, pp. 3887-3900, 024.
[8] A. Ganne, "Cloud data security methods: Kubernetes vs Docker swarm," *International Research J. of Modernization in Engineering Technology*, vol. 4, no. 11, pp. 1-6, 2022.
[9] G. Li, *et al.*, "The convergence of container and traditional virtualization: strengths and limitations," *SN Computer Science*, vol. 4, no. 4, Article ID: 387, 2023.
[10] M. Sobieraj and D. Kotyński, "Docker performance evaluation across operating systems," *Applied Sciences*, vol. 14, no. 15, Article ID: 6672, 2024.
[11] W. Shen, *et al.*, "Towards understanding and defeating abstract resource attacks for container platforms," *IEEE Trans. on Dependable and Secure Computing*, vol. 22, no. 1, pp. 474-490, Jan./Feb. 2024.
[12] D. Pennino and M. Pizzonia, Toward Scalable Docker-Based Emulations of Blockchain Networks for Research and Development, arXiv preprint arXiv:2402.14610, 2024.
[13] S. A. Baker, H. H. Mohammed, and O. I. Alsaif, "Docker container security analysis based on virtualization technologies," *International J. for Computers & Their Applications*, vol. 31, no. 1, pp. 69-78, 2024.
[14] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for high performance computing systems: survey and prospects," *IEEE Trans. on Software Engineering*, vol. 49, no. 4, pp. 2722-2740, Apr. 2022.
[15] N. Singh, *et al.*, "Load balancing and service discovery using Docker Swarm for microservice based big data applications," *J. of Cloud Computing*, vol. 12, Article ID: 4, 2023.
[16] S. T. Arzo, *et al.*, "Softwarized and containerized microservices-based network management analysis with MSN," *Computer Networks*, vol. 254, Article ID: 110750, Dec. 2024.
[17] O. I. Alqaisi, A. Ş. Tosun, and T. Korkmaz, "Performance analysis of container technologies for computer vision applications on edge devices," *IEEE Access*, vol. 12, pp. 41852-41869, 2024.
[18] D. Silva, J. Rafael, and A. Fonte, "Toward optimal virtualization: an updated comparative analysis of docker and LXD container technologies," *Computers*, vol. 13, no. 4, Article ID: 94, Apr. 2024.
[19] S. Tarasiuk, D. Traczuk, K. Szczepaniuk, P. Stoń, and J. Smołka, "Performance evaluation of designated containerization and virtualization solutions using a synthetic benchmark," *J. of Computer Sciences Institute*, vol. 32, pp. 157-162, 2024.
[20] D. P. VS, S. C. Sethuraman, and M. K. Khan, "Container security: precaution levels, mitigation strategies, and research perspectives," *Computers & Security*, vol. 135, Article ID: 103490, Dec. 2023.
[21] K. Senjab, S. Abbas, N. Ahmed, and A. U. R. Khan, "A survey of Kubernetes scheduling algorithms," *J. of Cloud Computing*, vol. 12, Article ID: 87, 2023.
[22] E. Truyen, H. Xie, and W. Joosen, "Vendor-agnostic reconfiguration of kubernetes clusters in cloud federations," *Future Internet*, vol. 15, no. 2, Article ID: 63, Feb. 2023.
[23] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões, "Container-based virtualization for real-time industrial systems-a systematic review," *ACM Computing Surveys*, vol. 56, no. 3, Article ID: 59, Mar. 2023.
[24] A. Alamoush and H. Eichelberger, "Open source container orchestration for industry 4.0-requirements and systematic feature analysis," *International J. on Software Tools for Technology Transfer*, vol. 26, pp. 527-550, 2024.
[25] O. Flauzac, F. Mauhourat, and F. Nolot, "A review of native container security for running applications," *Procedia Computer Science*, vol. 175, pp. 157-164, 2020.
[26] S. Kaiser, M. S. Haq, A. Ş. Tosun, and T. Korkmaz, "Container technologies for arm architecture: a comprehensive survey of the state-of-the-art," *IEEE Access*, vol. 10, pp. 84853-84881, 2022.
[27] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-Centric Computing and Information Sciences*, vol. 8, Article ID: 63, 15 pp., 2018.
[28] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: application, orchestration and security," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, Article ID: e5668, Sept. 2020.
[29] D. Moreau, K. Wiebels, and C. Boettiger, "Containers for computational reproducibility," *Nature Reviews Methods Primers*, vol. 3, no. 1, p. 50, 2023.
[30] V. P. M. John, "A study on cloud container technology," *i-Manager's J. on Cloud Computing*, vol. 10, no. 1, pp. 7-16, Jan./Jun. 2023.
[31] H. Liu, W. Zhu, S. Fu, and Y. Lu, "A trend detection-based auto-scaling method for containers in high-concurrency scenarios," *IEEE Access*, vol. 12, pp. 71821-71834, 2024.
[32] Z. P. Putro and R. A. Supono, "Comparison analysis of apache and Nginx webserver load balancing on proxmox VE in supporting server performance," *International Research J. of Advanced Engineering and Science*, vol. 7, no. 3, pp. 144-151, 2022.
[33] C. T. Yeh, T. M. Chen, and Z. J. Liu, "Flexible IoT cloud application for ornamental fish recognition using YOLOv3 model," *Sensors & Materials*, vol. 34, no. 3, pp. 1229-1240, 2022.
[34] M. Kwon, *et al.*, "Deterministic I/O and resource isolation for OS-level virtualization in server computing." in *Proc. 12th Annual Non-Volatile Memories Workshop,* 2 pp., 7-21 Mar. 2021
[35] D. Šandor and M. Bagić Babac, "Designing scalable event-driven systems with message-oriented architecture," *Distributed Intelligent Circuits and Systems,* Ch. 2, pp. World Scientific, 2024.
[36] D. DeJonghe, *Nginx Cookbook*, O'Reilly Media, 2020.

علی فرهادیان در سال ۱۳۹۸ مدرک کارشناسی مهندسی کامپیوتر خود را از دانشگاه صنعتی ارومیه و در سال ۱۴۰۲ مدرک کارشناسی ارشد مهندسی کامپیوتر خود را از دانشگاه مازندران دریافت نمود. از سال ۱۴۰۱ تا کنون نام‌برده در تعدادی از شرکت‌های فعال در حوزه فناوری اطلاعات به‌عنوان کارشناس زیرساخت و DevOps به کار مشغول بوده است. زمینه‌های علمی مورد علاقه مهندس فرهادیان شامل موضوعاتی مانند توسعه و مقیاس‌پذیری زیرساخت‌های ابری، نیمه ابری و شبکه‌های توزیع‌شده است.

مصطفی بستام استادیار دانشکده مهندسی کامپیوتر دانشگاه مازندران است. او دکترای خود را در رشته شبکه‌های کامپیوتری از دانشگاه صنعتی امیرکبیر (پلی‌تکنیک تهران)، ایران، در سال ۱۳۹۶ دریافت کرد. مدرک کارشناسی خود را در رشته مهندسی کامپیوتر از دانشگاه شهید باهنر کرمان، ایران، در سال ۱۳۸۵ و مدرک کارشناسی ارشد را در رشته مهندسی کامپیوتری از دانشگاه صنعتی امیرکبیر، ایران، در سال ۱۳۸۸ دریافت کرد. زمینه‌های علمی مورد علاقه نام‌برده شامل موضوعاتی مانند رایانش ابری، شبکه‌های نرم‌افزار محور، اینترنت اشیا، یادگیری ماشین و بلاکچین است.

احسان عطائی تحصیلات خود را در مقاطع کارشناسی، کارشناسی ارشد و دکتری رشته مهندسی کامپیوتر، به ترتیب در ساله‌ای ۱۳۸۱، ۱۳۸۳ و ۱۳۹۶ از دانشگاه صنعتی شریف تهران اخذ نموده و هم اکنون دانشیار گروه مهندسی کامپیوتر دانشکدهٔ مهندسی و فناوری دانشگاه مازندران است. زمینه‌های پژوهشی مورد علاقه ایشان شامل رایانش ابری، سیستم‌های توزیع شده، مدل‌سازی کارایی و اتکاپذیری است.

مهدی باباگلی مدرک کارشناسی خود را در رشته مهندسی فناوری اطلاعات در سال ۱۳۹۳ از دانشگاه مازندران اخذ نموده و سپس در سال ۱۳۹۵ کارشناسی ارشد خود را در دانشگاه صنعتی ارومیه به پایان رسانده است. وی مدرک دکتری خود را در سال ۱۴۰۲ از دانشگاه خواجه نصیرالدین طوسی دریافت کرده است و هم اکنون عضو هیات علمی دانشگاه مازندران، دانشکده مهندسی کامپیوتر می باشد. مرتبه علمی ایشان استادیاری بوده وزمینه فعالیت ایشان امنیت شبکه های کامپیوتری و علوم داده می‌باشد.