DOR :20.1001.1.2538161.1402.8.1.2.4

بهار ۱۴۰۲، دوره ۸، شماره ۱، پیاپی ۲۸

دریافت مقاله: ۱۴۰۱/۰۸/۲۷
پذیرش مقاله: ۱۴۰۱/۱۱/۲۶

نوع مقاله: پژوهشی

کاهش شروع سرد در رایانش بی‌خدمت‌گزار با بهبود الگوریتم زمان‌بندی و مدیریت منابع

محمدامین چینی‌فروشان اصفهانی
دانشجوی کارشناسی ارشد دانشکده کامپیوتر ـ دانشگاه علم و صنعت ـ تهران ـ ایران
پست الکترونیکی: m_chiniforoushan@comp.iust.ac.ir

مهرداد آشتیانی*
استادیار دانشکده کامپیوتر ـ دانشگاه علم و صنعت ـ تهران – ایران
پست الکترونیکی: m_ashtiani@iust.ac.ir

فاطمه بخشی
دانشجوی دکتری دانشکده کامپیوتر ـ دانشگاه علم و صنعت ـ تهران – ایران
پست الکترونیکی: fatemeh_bakhshi@comp.iust.ac.ir

چکیده

رایانش بی‌خدمت‌گذار، یک روش تامین منابع موردنیاز برای نیازهای پردازشی است. بر خلاف آنچه از اسم این روش برمی‌آید، در این روش از خدمت‌گذار استفاده می‌شود، اما اصطلاح بی‌خدمت‌گذار به این دلیل به این نوع پردازش داده می‌شود که هزینهٔ استفاده از آن با توجه به میزان استفاده واقعی از خدمت‌گذار محاسبه می‌شود و یک مقدار مشخص برای یک بازه زمانی نیست. از مزایای این مدل رایانشی می‌توان به کاهش هزینه و مقیاس‌پذیری آسان اشاره کرد. ارائه این مزایا باید بدون افت کیفیت و سرعت بُن‌سازه باشد. بُن‌سازه با دریافت درخواست‌ها، محیط‌های عملکردی جدیدی را فعال می‌کند که اصطلاحاً به این راه‌اندازی، شروع سرد می‌گوییم. تأخیر در راه‌اندازی می‌تواند منجر به تاخیر در پاسخ‌دهی به درخواست و حتی شکست درخواست شود. همچنین فعال‌سازی محیط‌های عملکردی اگر هوشمندانه صورت نگیرد می‌تواند منابع پردازشی زیادی را هدر دهد. در این پژوهش روشی مناسب برای مدیریت منابع پردازشی و کاهش زمان و تعداد رخداد شروع سرد در رایانش بی‌خدمت‌گذار با بهبود الگوریتم زمان‌بندی فعال‌سازی محیط‌های عملکردی ارائه شده است. روش ارائه‌شده در زمان ورود درخواست، منابعی از بُن‌سازه که در حال میزبانی از یک تابع درحال اجرا هستند را بررسی می‌کند و سپس زمان انتظار برای منبعی که بالاترین تشابه با درخواست ورودی را دارد، محاسبه می‌شود. اگر زمان انتظار کمتر از زمان لازم برای اختصاص منبع جدید باشد، بُن‌سازه تصمیم می‌گیرد که درخواست منتظر آزاد شدن منبع مشابه بماند. نتایج به‌دست‌آمده از دو آزمایش طراحی شده حاکی از آن است که این روش زمان پاسخ را در شرایط عادی ترافیک درخواست ورودی تا ۱۰ درصد و در شرایط شلوغی درخواست‌های ورودی، تا ۱۶ درصد نسبت به حالتی که بر بُن‌سازه سیاستی برای مدیریت درخواست‌ها اعمال

* نویسندهٔ مسئول

نشود، کاهش می‌دهد. این روش همچنین در شرایط عادی ترافیک درخواست ورودی تا ۱۳ درصد و در شرایط شلوغی تا ۳۳ درصد در مصرف منابع نسبت به حالتی که بر بُن‌سازه سیاستی برای مدیریت درخواست‌ها اعمال نشود، صرفه‌جویی می‌کند.

واژه‌های کلیدی: شروع سرد، رایانش بی‌خدمت‌گذار، تابع به‌عنوان خدمت، زمان‌بندی.

۱. مقدمه

رایانش بی‌خدمت‌گذار[1] به معنای نیاز نداشتن به خدمت‌گذار نیست، بلکه به معنای نداشتن نگرانی در مورد پیکربندی و مدیریت خدمت‌گذار توسط توسعه‌دهنده برنامه کاربردی است. این مدل رایانشی، معماری نرم‌افزار تابع به عنوان خدمت[2] را مطرح می‌کند که طبق آن یک برنامه کاربردی به واحدهایی مستقل به نام تابع تجزیه می‌شود. طبق این مدل رایانشی، توسعه‌دهنده فقط بر منطق برنامه و نوشتن توابع تمرکز می‌کند و مسئولیت هر گونه نیاز به پیکربندی، مدیریت خدمت‌گذارها، منابع و پس‌زمینه برنامه‌های کاربردی بر عهده فراهم‌کننده ابر است [۱]. یکی از مهم‌ترین چالش‌های رایانش بدون خدمت‌گذار، چالش شروع سرد[3] است که از قابلیت مقیاس‌پذیری به صفر[4] و مقیاس‌پذیری افقی[5] در این مدل رایانشی ناشی می‌شود. قابلیت مقیاس‌پذیری به صفر منجر به بهبود هزینه می‌گردد. زیرا طبق این قابلیت، با اتمام اجرای یک تابع، تمام منابع از آن پس گرفته می‌شود. بنابراین، در زمان‌هایی که توابع غیرفعال هستند، هزینه‌ای دریافت نمی‌شود. اما برای پاسخ به درخواست‌های آینده، باید از ابتدا منابع به تابع اختصاص داده شوند تا تابع آماده اجرا شود که این مراحل مدت زمانی طول می‌کشد که با عنوان شروع سرد شناخته می‌شود [۲]. قابلیت مقیاس‌پذیری افقی نیز در مواقعی که ترافیک درخواست‌های ورودی به بُن‌سازه افزایش می‌یابد، منابع جدیدی برای اجرای توابع اختصاص می‌دهد و این باعث می شود این درخواست‌ها با شروع سرد مواجه شوند . روش‌هایی که تا کنون ارائه شده‌اند غالباً مبتنی بر استفاده از محیط‌های اجرایی گرم هستند و همگی سعی دارند تا تاثیرات نامطلوب شروع سرد، ازجمله تاخیر درپاسخ‌دهی به درخواست‌های ورودی به بُن‌سازه، شکست درخواست ورودی به بُن‌سازه و مصرف غیربهینه منابع پردازشی بُن‌سازه که می‌تواند منجر به افزایش هزینه‌های مشتریان شود را بهبود دهند. دسته‌ایی از روش‌هایی که تاکنون ارائه شده‌اند سعی بر فعال نگه داشتن محیط اجرایی پس از اتمام کار تابع اجرا شده در آن محیط دارند و از روش‌های مختلفی همچون پیش‌بینی درخواست تابع استفاده می‌کنند. در این روش‌ها که مبتنی بر استفاده از محیط اجرایی گرم هستند، به این نکته که محیط‌های اجرایی گرمی که هم‌اکنون در حال اجرای یک تابع هستند و توانایی پذیرش درخواست ورودی جدید را دارند، توجه نمی‌شود. با توجه به این نکته، در این پژوهش یک روش جدید برای کاهش دفعات رخداد شروع سرد و کاهش زمان رخداد شروع سرد در رایانش بی‌خدمت‌گذار ارائه شده است. روش ارائه شده که از آن با نام سیاست انتظار در بُن‌سازه تابع به‌عنوان خدمت نیز یاد می‌شود، در هنگام ورود درخواست با بررسی منابع موجود در بُن‌سازه و در صورت وجود منبع مناسب برای اجرای درخواست، زمان‌بندی اجرای درخواست را تغییر داده و منتظر می‌ماند که منبع مناسب انتخاب شده کارش به اتمام برسد. البته شرط مهمی که باید درنظر گرفته شود این است که این انتظار از زمان لازم برای متحمل شدن شروع سرد کم‌تر باشد. با به کارگیری این روش نه تنها تعداد رخداد و زمان شروع سرد کاهش می‌یابد، بلکه در مصرف منابع پردازشی بُن‌سازه صرفه‌جویی می‌شود. انتظار می‌رود که این روش در زمان‌های شلوغی بُن‌سازه، زمان پاسخ‌دهی به درخواست‌های ورودی را نسبت به یک بُن‌سازه که در آن سیاستی برای زمانبندی این‌گونه درخواست‌ها حاکم نیست، به‌طور میانگین کم‌تر

1- Serverless Computing
2- Function as a Service(FaaS)
3- Cold Start
4- Zero Scaling
5- Horizontal Scaling

کند و کارایی سامانه را افزایش دهد. برای ارزیابی روش ارائه شده دو آزمایش با مدت اجرای ۳۶۰۰ واحد زمانی (معادل یک ساعت) طراحی شده است. در آزمایش اول درخواست‌ها با نرخ ثابت وارد می‌شوند و در آزمایش دوم نرخ ورود درخواست‌ها ثابت نبوده و دارای یک اوج در ترافیک ورودی است تا نتیجه اعمال روش در زمان‌های شلوغی بُن‌سازه بررسی شود. معیارهای ارزیابی این دو آزمایش، بهبود زمان پاسخ، کاهش رخداد شروع سرد و بهبود مصرف منابع هستند تا بتوان میزان کارآمدی روش را برای بهبود تاثیرات نامطلوب یاد شده سنجید. ادامه این مقاله، در بخش ۲ مفاهیم پایه شرح داده می‌شود، در بخش ۳ پژوهش‌های مرتبط مورد بررسی قرار می‌گیرد. رویکرد پیشنهادی در بخش ٤ معرفی شده و ارزیابی رویکرد پیشنهادی در بخش ٥ انجام می‌شود. در نهایت بخش ٦ شامل نتیجه‌گیری است.

۲. مفاهیم اولیه

به منظور معرفی رایانش بی‌خدمت‌گذار ابتدا مفاهیم کلی مربوط به رایانش ابری شرح داده می‌شود. سپس مجازی‌سازی که یکی از مهم‌ترین فناوری‌های رایانش ابری است معرفی می‌شود. در انتها به خدمت و رایانش بی‌خدمت‌گذار پرداخته شده است.

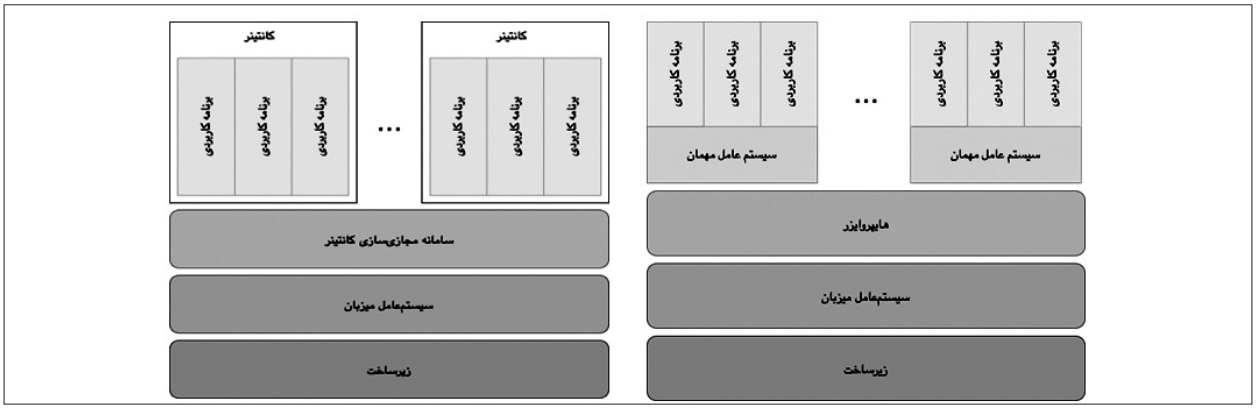
۲-۱ رایانش ابری و مدل‌های خدمت

رایانش ابری تحولی در فناوری اطلاعات و یک مدل تجاری غالب برای ارائه منابع فناوری اطلاعات است. با رایانش ابری، افراد و سازمان‌ها می‌توانند به‌صورت درخواست- محور به شبکه مشترکی از منابع فناوری اطلاعات مدیریت‌شده و مقیاس‌پذیر مانند خدمتگذارها، فضای ذخیره‌سازی و برنامه‌ها دسترسی پیدا کنند. امروزه بسیاری از شرکت‌ها و کاربران در زندگی روزمره خود به خدمات ابری متکی هستند. به عنوان مثال، برای ذخیره داده‌ها، نوشتن اسناد، مدیریت کسب وکار و بازی‌های برخط از این خدمات استفاده می‌شود. رایانش ابری همچنین زیرساختی را فراهم می‌کند که به روندهای دیجیتالی کلیدی مانند محاسبات تلفن همراه، اینترنت اشیاء کلان‌داده‌ها و هوش مصنوعی کمک می‌کند و در نتیجه پویایی صنعت را تسریع می‌بخشد [۳]. این مدل رایانشی دارای سطوح مختلفی از انتزاع است، در پایین‌ترین سطح انتزاع مدل رایانشی زیرساخت به‌عنوان خدمت[6] مطرح می‌شود که طبق آن ماشین‌های مجازی به سازمان‌ها ارائه می‌شوند. با این روش می‌توان چندین ماشین مجازی را روی یک خدمت‌گذار اجرا کرد. در سطح بالاتر، بُن‌سازه به‌عنوان خدمت[7] قرار دارد که طبق آن یک محیط توسعه برنامه‌کاربردی به توسعه‌دهندگان ارائه می‌گردد و در بالاترین سطح، نرم‌افزار به‌عنوان خدمت[8] قرار دارد که در آن کل یک سیستم نرم‌افزاری در قالب خدمت توسط ارائه‌دهنده ابر میزبانی می‌شود. جدیدترین مدل خدمت در رایانش ابری، معماری تابع به عنوان خدمت[9] است که از آن با نام رایانش بدون خدمت‌گذار نیز یاد می‌شود [۲].

۲-۲ مجازی‌سازی

مجازی‌سازی عملی است که در طی آن منابع فیزیکی مانند سخت‌افزارهای رایانشی، فضای ذخیره‌سازی و یا منابع شبکه به طور مجازی در اختیار کاربر قرار می‌گیرد. نرم‌افزارهای مجازی‌سازی به عنوان یک لایه میانی بین سخت‌افزار و منابع فیزیکی قرار می‌گیرند و آن‌ها را با تقسیم‌بندی‌های خودکار و یا دستی بین نرم‌افزارهای سطح بالاتر خود توزیع می‌کنند. مجازی‌سازی می‌تواند با استفاده از ماشین مجازی انجام شود. در این معماری، برنامه‌کاربردی روی یک سیستم‌عامل میهمان که خود روی یک سیستم‌عامل میزبان قرار دارد، قرار می‌گیرد. ماشین‌های مجازی یک برابرساز[10] از رایانه‌های فیزیکی هستند. این برابرسازی با استفاده از برنامه‌های فراناظر[11] انجام می‌شود که در لایه بالای سیستم‌عامل میزبان و به منظور خدمت‌رسانی به سیستم‌عامل‌های میهمان

6- Infrastructure as a Service
7- Platform as a Service
8- Software as a Service
9- Function as a service
10- Emulator
11- Hypervisor



شکل ۱: معماری ماشین‌های مجازی (شکل راست) و کانتینر (شکل چپ)

قرار می‌گیرد [٤]. کانتینرها[12] روشی دیگر برای رسیدن به هدف مجازی‌سازی است که در سطح سیستم‌عامل عمل می‌کنند. در این روش، برخلاف ماشین مجازی، به جای چندین سیستم‌عامل میهمان، فراناظر و سیستم‌عامل میزبان، از یکی از فناوری‌های مجازی‌سازی با کانتینر برروی سیستم‌عامل میزبان بهره می‌گیرد. این فناوری‌های مجازی‌سازی (مانند داکر[13]) اجرای نرم‌افزارهای مختلف را از یکدیگر ایزوله می‌کند. بنابراین، روش استفاده از کانتینر روشی سبک‌تر از ماشین‌های مجازی محسوب می‌شود [٥]. در شکل (۱) تفاوت این دو معماری نشان داده شده است.

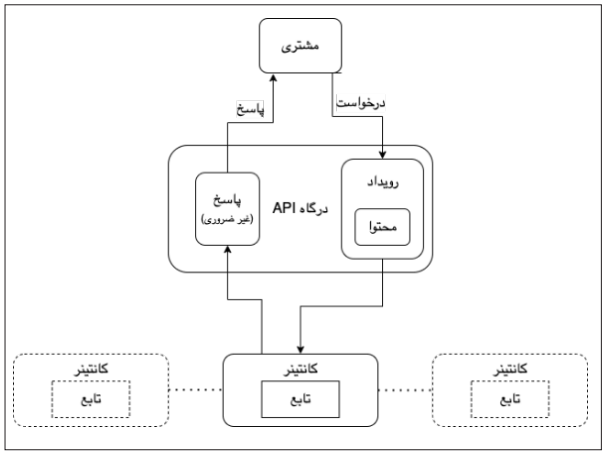
۲-۳ خدمت

هر سازوکاری که ما را قادر به دسترسی یک یا چند قابلیت کند، به طوری که این دسترسی از طریق یک رابط و مطابق با محدودیت‌ها و سیاست‌هایی از پیش‌تعریف شده باشد را خدمت می‌گوییم. این سازوکار می‌تواند یک برنامه و یا تابع نرم‌افزاری باشد [٦]. معماری خدمت‌گرا[14] روشی از طراحی نرم‌افزاری است که در آن اجزای یک برنامه به واسطه یک پروتکل ارتباطی در شبکه به سایر اجزا خدمات ارائه می‌دهند. یک خدمت در معماری خدمت‌گرا، یک واحد عملیاتی است که می‌توان از راه دور به آن دسترسی داشت و به طور مستقل آن را به روزرسانی کرد [٧]. رایانش خدمت‌گرا[15] الگوی رایانشی جدید است که از خدمت‌ها به عنوان سازه‌هایی ابتدایی برای توسعه سریع، کم‌هزینه و آسان نرم‌افزارهای توزیع‌شده حتی در محیط‌های همگن بهره می‌برد. هدف از این مدل رایانشی تحقق جهانی است که در آن خدمت‌های مختلف برای ساخت نرم‌افزارها دست به دست هم می‌دهند به‌طوری که برای ایجاد فرآیندهای پویا، انعطاف‌پذیر و برای برنامه‌های کاربردی سازمان‌ها چابکی ممکن را داشته باشند [٨]. یک خدمت خرد، یک جز مستقل قابل استقرار با مرز و محدوده مشخص است که از ارتباطات به واسطه پروتکل‌های مبتنی بر رد و بدل کردن پیام پشتیبانی می‌کند. معماری خدمات‌خرد یکی از روش‌های مهندسی سامانه‌های نرم‌افزاری قابل ارتقا و تا حد زیادی خودکار است که از خدمات‌خرد با قابلیت‌های تراز شده استفاده می‌کند [٩].

۲-٤- رایانش بی‌خدمت‌گذار

این مدل رایانشی منجر به افزایش ریزدانگی برنامه‌های کاربردی می‌شود. به‌طوری که طبق آن، یک برنامه کاربردی به واحدهایی مستقل به نام تابع که یک قطعه کد کوچک، قابل اجرا و تک‌وظیفه‌ای است تجزیه می‌شود. در شکل (۲) جزئیات مدل رایانش بی‌خدمت‌گذار مشاهده می‌شود، این معماری مبتنی بر رویداد است. یعنی برای اجراشدن یک تابع باید رویدادی رخ‌دهد تا آن تابع فراخوانی[16] شود. یک رویداد عبارت است از درخواست‌های وارد شده توسط واسط برنامه کاربردی، مثل فراخوانی سیستم احراز

12- Container
13- Docker
14- Service Oriented Architecture (SOA)
15- Service Oriented Computing (SOC)
16- Trigger



شکل ۲: فعال‌شدن اجرای تابع توسط مشتری

هویت. یک رویداد محیطی سبب فراخوانی و اجرای یک یا چند تابع می‌شود و توابع نیز ممکن است برای پاسخ به درخواست ورودی، یکدیگر و یا سایر خدمت‌های موجود در اکوسیستم ابری را فراخوانی کنند. با فراخوانی هر تابع، بُن‌سازه برای اجرای آن یک کانتینر اختصاص می‌دهد.

۳- پژوهش‌های مرتبط

طبق مطالعات انجام شده، دو رویکرد کلی برای کاهش تاخیر شروع سرد وجود دارد. هدف رویکرد اول به حداقل رساندن مدت زمان تاخیر و هدف رویکرد دوم به حداقل رساندن بسامد رخداد شروع سرد است. در ادامه پژوهش‌های مرتبط بر اساس این دسته‌بندی معرفی می‌شوند.

۳-۱- راه‌حل‌های ارائه‌شده برای به حداقل رساندن مدت زمان تاخیر شروع سرد

راه‌حل‌های ارائه‌شده برای این رویکرد را می‌توان به دو دسته تقسیم‌کرد. دسته اول راه‌حل‌های ارائه‌شده برای کاهش تاخیر آماده‌سازی کانتینر و دسته دوم راه‌حل‌های ارائه‌شده برای کاهش تاخیر بارگذاری کتابخانه‌های مورد نیاز توابع است که در ادامه به شرح آن‌ها خواهیم پرداخت.

۳-۱-۱ راه‌حل‌های ارائه‌شده برای کاهش تاخیر آماده‌سازی کانتینر

بُن‌سازه متن‌باز اُپِن‌فَس[17] از یک کانتینر همواره درحال اجرا به ازای هر تابع استفاده می کند. گرچه در این روش مشکل شروع سرد تا حدی رفع می‌شود، ولی منابع زیرساختی بسیار زیادی را مصرف می‌کند و باعث افزایش هزینه‌های توسعه‌دهندگان این برنامه‌کاربردی می‌شود [۱۰]. در بُن‌سازه‌های اُپِن‌ویسک[18] و ای‌دبلیواس[19] اولین اجرای یک تابع متحمل تاخیر شروع سرد می‌گردد. اما بعد از پایان اجرا، کانتینر قبل از آزاد شدن به حالت توقف می‌رود. اگر در این بازه زمانی درخواست‌هایی وارد شوند، کانتینر از حالت توقف خارج شده و دوباره مورد استفاده قرار می‌گیرد. این روش می‌تواند برای کاهش تاخیر موثر باشد اما منجر به اشغال حافظه و هدر رفتن منابع می‌گردد [۱۰, ۱۱]. بُن‌سازه‌های فیشِن[20] و نِیتیو[21] استخری از کانتینرهایی همیشه درحال اجرا دارند که منجر به کاهش تاخیر و افزایش کارایی می‌شود اما از نظر مصرف منابع و افزایش هزینه به صرفه نیست [۱۲]. بُن‌سازه‌های گوگل[22]، ای‌دبلیواس و آزور[23] پس از اتمام اجرای یک تابع، کانتینر را برای پاسخ به درخواست‌های احتمالی بعدی حفظ می‌کند. هر کدام از این بُن‌سازه‌ها سیاست متفاوتی برای میزان گرم نگه‌داشتن کانتینرها دارند. بُن‌سازهٔ آزور نسبت به دو بُن‌سازه قبلی هوشمندانه‌تر عمل‌می‌کند و باتوجه به تاریخچه فراخوانی‌ها مدت زمان متفاوتی را برای گرم نگه‌داشتن کانتینرها، در نظر می‌گیرد [۱۳]. گرت و همکاران [۱۴] بُن‌سازه‌ای برای رایانش بی‌خدمت‌گذار ارائه کرده‌اند که شامل یک صف از کانتینرهای گرم و در دسترس به ازای هر تابع و یک صف عمومی از کانتینرهای سرد به ازای تمام توابع است که هنوز حافظه به آن‌ها تعلق نگرفته است. این بُن‌سازه ابتدا از کانتینرهای موجود در صف گرم استفاده می‌کند و در

17- OppenFaas
18- OpenWhisk
19- AWS Lambda
20- Fission) https://fission.io/docs/ (
21- Knative
22- Google Cloud Functions
23- Azure Functions

صورت در دسترس نبودن کانتینر گرم، یک کانتینر موجود در صف سرد به تابع اختصاص داده می‌شود. در این روش هـم به دلیل وجود صفی آماده از کانتینرهای گرم، مصرف منابع بهینه نیست. راه‌حل بُن‌سازه ارائه شده توسط ایستمی و همـکاران [۱۵] برای کاهش تاخیر، جداسـازی منطق در سـطح برنامه کاربردی است. به این صورت به جای این‌که توابع یک برنامه کاربردی در کانتینرهای مسـتقلی مستقر شـوند، هر برنامه کاربردی روی یک کانتینر قرار می‌گیرد و توابع آن توسط پردازش‌های مختلف در سطح یک کانتینر از هم جدا می‌شـوند. با استفاده از این رویکرد اجرای توابع با ایجاد یک پردازش جدید، تاخیر کمتری نسـبت به ایجاد یک کانتینر جدید دارد و کتابخانه‌هایی که بین توابع مشترک هسـتند، فقط یک بار روی کانتینر بارگذاری می‌شـوند. در صورتی که تعداد توابع یک برنامه کاربردی زیاد باشد و یا بیشتر مولفه‌های یک برنامه کابردی به‌صورت تابع به عنوان خدمت توسعه یافته‌باشند، استفاده از منابع بهینه می‌شود. اما معمولاً فقط بعضی از مولفه‌ها قابلیت تجزیه شدن به توابع را دارند. پریچهر و همکاران [۱۶] یک رویکرد تطبیقی دو لایه جدید برای مقابله با این موضوع پیشنهاد کرده‌اند که لایه اول از یک الگوریتم یادگیری تقویتی جامع برای کشف الگوهای فراخوانـی تابع در طـول زمان برای تعییـن بهترین زمان برای گرم نگه داشتن کانتینرها استفاده می‌کند. لایه دوم بر اسـاس یک حافظه طولانی-کوتاه مدت[۲۴] طراحی شده است تـا زمان‌های فراخوانی تابع را در آینده برای تعیین کانتینر از قبل گرم شده مورد نیاز پیش‌بینی کند. براساس تحقیقات آن‌ها که برروی بُن‌سـازهٔ اُپن‌ویسک انجام شده است، این روش میزان مصـرف حافظه را تـا ۱۲/۳۷ درصد کاهش می‌دهد و همچنین منجر به کاهش ۲۲/۶۵ درصدی در زمان اجرای توابع نسبت به سیاسـت عادی بُن‌سازه اُپن‌ویسک می‌شـود. دانیال و همکاران [۱۷] روش‌هایی ارائه کردند که با اعمال تغییر در تصاویـر پایه[۲۵]، نحوه بارگذاری ورودی خروجی و نحوه ساخت پایگاه داده، تاخیر شروع سرد را تا ۱۰/۵ درصد برای توابعی که وابسـته به پایگاه داده هستند کاهش می‌دهد. پاولو و همکاران [۱۸] روشی ارائه کردند که بُن‌سـازه برای اجرای مجدد یک تابع از تصویر لحظه‌ای[۲۶] اجرای قبلی تابع اسـتفاده می‌کند. آزمایش‌های آن‌ها نشان می‌دهد که زمانی که بُن‌سازه تصمیم می‌گیرد که چه زمانی از اجرای تابع تصویر لحظه‌ایی را بسـازد تاثیر بسزایی در استفاده مجدد از این تصویر در اجراهای بعدی خواهد داشت تا جایی که می‌تواند زمان اجرا را تا ٤ برابر کاهش دهد.

۳-۱-۲ راه‌حل‌های ارائه‌شده برای کاهش تاخیر بارگذاری کتابخانه‌های مورد نیاز توابع

در برخی روش‌ها بر اساس میزان استفاده، کتابخانه‌هایی نگه‌داری می‌شوند. مانند راه حل استفاده شده توسط ایستمی و همکاران [۱۵] که با بررسـی رفتار و تاریخچه کانتینرها، کتابخانه‌های مهم را قبل از اجرای توابع نصب و روی حافظه دیسک نگه می‌دارد. راه‌حل بعدی جداسازی منطق در سطح برنامه‌کاربردی است که منجر به کاهش تاخیر آماده‌سازی کانتینـر می‌شـود. همچنین به این دلیل که تمـام توابع یک برنامه‌کاربـردی روی یک کانتینر قـرار می‌گیرند، به ازای تمام توابعی که از یک کتابخانه مشترک استفاده می‌کنند، این بارگذاری فقط یک بار انجام می‌شـود. اما در سایر روش‌ها که هر تابع روی کانتینر مجزا مسـتقر می‌شود، بارگذاری مجددکتابخانه‌ها صورت می‌گیـرد. آکآش و همکاران [۱۹] از سـاریما[۲۷] که یکی از مدل‌های سنتی پیش‌بینی سری‌های زمانی است استفاده کرده‌اند تا زمان ورود دروخواست‌ها به بُن‌سازه را پیش‌بینی کنند و با توجه به آن تعداد کانتینرهای فعال بُن‌سازه را افزایش یا کاهش‌دهند و بدین صورت هم از تاخیر شروع سرد و هم از اتلاف منابع پردازشی جلوگیری کنند.

۳-۲ راه‌حل‌های ارائه‌شده برای به حداقل رساندن فرکانس رخداد شروع سرد

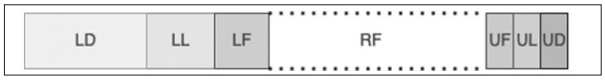
یکـی از راه‌حلی‌هایـی که برای کاهش شـروع سـرد برطبق رویکرد به حداقل رسـاندن بسـامد رخداد شروع

24- Long Short-Term Memory (LSTM)
25- Base Images
26- Snapshot
27- Seasonal Auto Regressive Integrated Moving Average (SARIMA)

سرد ارائه شده است، فراخوانی دوره‌ای توابع است. ابزارهای کلادواچ[۲۸]، توندرا[۲۹] و داشبرد[۳۰] برای تنظیم قوانین و کارها به منظور فراخوانی دوره‌ای توابع استفاده می‌شوند. فراخوانی دوره‌ای توابع از آزاد شدن منابع و سرد شدن تابع جلوگیری می‌کند و رخداد شروع سرد را کاهش می‌دهد. ابزار وارمر [۳۱]، یک زیربرنامه[۳۲] سبک وزن است که می‌تواند به توابع اضافه شده تا رویدادها را برای گرم نگه‌داشتن توابع مدیریت کند. بیسواجیت و همکاران [۲۰] روشی برای کاهش رخداد شروع شروع سرد ارائه کرده‌اند که در آن کانتینرها را برای مدت طولانی تری فعال نگه می‌دارند و این فعال نگه داشتن بر اساس کانتینرهایی است که کمترین مورد استفاده اخیر را داشته‌اند. نتایج این روش نشان داده است که پیشرفت ٤٨ درصدی در کاهش رخداد شروع سرد در مقایسه با روش انتخاب کانتینر با بیشترین استفاده اخیر صورت گرفته است.

۴- روش پیشنهادی

در روش ارائه‌شده ، هدف، کاهش زمان پاسخ به درخواست‌های ورودی به بُن‌سازه، در زمان‌هایی است که تعداد زیادی درخواست در حال ورود به بُن‌سازه هستند و منابع پردازشی زیادی به کار گرفته شده‌اند تا به این درخواست‌ها پاسخ داده شود. به طور کلی اگر بُن‌سازه سیاست خاصی برای مدیریت نداشته باشد، به ازای هر درخواست ورودی یک کانتینر می‌سازد تا تابع پاسخ‌گو به آن درخواست را درون کانتینر اجرا کند و درنهایت پس از پاسخ به درخواست، کانتینر ساخته شده را حذف می‌کند. نکته مهمی که باید درنظر گرفته شود این است که بخش قابل توجهی از زمان فعالیت کانتینرها، شامل شروع و راه‌اندازی محیط اجرایی شده و همچنین بخشی از آن‌هم صرف نهایی‌سازی و ارسال اطلاعات اجرا به



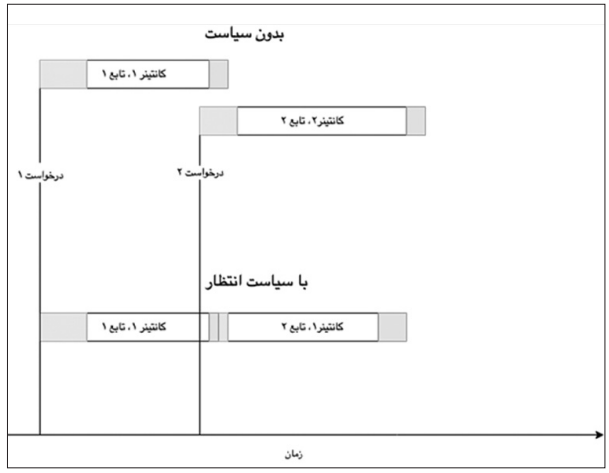
شکل ۳: جزئیات چرخه حیات یک کانتینر

بُن‌سازهٔ ابری به منظور بهبود و یا پایش خدمت می‌شود. بنابراین، تمام زمان فعالیت کانتینر فقط صرف اجرای تابع مهمان نمی‌شود و بخشی از این زمان صرف فعال‌سازی و نهایی‌سازی توابع می‌شود. در شکل (۳) چرخه حیات یک کانتینر نشان داده شده است. این چرخه حیات از سه بخش کلی تشکیل شده است که هر بخش شامل چند قسمت است. این قسمت‌ها شامل فعال‌سازی (بارگذاری وابستگی‌های تابع[۳۳]، بارگذاری زبان اجرای تابع[۳٤] و بارگذاری بسته‌کد تابع[۳۵])، اجرای تابع[۳٦] و نهایی‌سازی (ارسال اطلاعات اجرا تابع به سکوی ابری و آزادسازی بسته‌کد بارگذاری‌شده[۳۷]، آزادسازی زبان اجرای تابع[۳۸] و آزادسازی وابستگی‌های تابع[۳۹]) است. بخش فعال‌سازی، بخشی است که از آن به عنوان «شروع سرد» یاد می‌شود. هر قسمت از بخش فعال‌سازی می‌تواند باتوجه درخواست ورودی به بُن‌سازه زمان مختلفی را صرف‌کند و زمان شروع سرد متفاوتی ایجادکند.

باتوجه به زمان‌های صرف‌شده برای ساخت و پایان‌دادن به یک کانتینر، به نظر می‌رسد که یک بُن‌سازه بدون سیاست در اوقاتی که حجم درخواست‌های ورودی به بُن‌سازه زیاد است، باید زمان و منابع زیادی را صرف یک درخواست کند و این درحالی است که ممکن است یک درخواست ورودی، در مراحل آماده‌سازی، شباهت زیادی با یکی از کانتینرهایی که در زمان ورود درخواست درحال اجرا هستند داشته‌باشد. بنابراین بُن‌سازهٔ دارای سیاست انتظار می‌تواند با توجه به اینکه چه مقدار از زمان اجرای تابع در کانتینر مشابه باقی مانده است، تصمیم بگیرد که درخواست ورودی منتظر تابع در کانتینر مشابه بماند و پس از اتمام کار تابع از منابع کانتینر فعال استفاده کند، یا

28- Cloud Watch https://aws.amazon.com/cloudwatch/
29- Thundra.io https://www.thundra.io/
30- Dashbird.io https://dashbird.io/
31- lambda Warmer https://github.com/jeremydaly/lambda-warmer
32- Module

33- Load Dependencies (LD)
34- Load Language (LL)
35- Load Function(LF)
36- Run Function (RF)
37- Unload Function (UF)
38- Unload language (UL)
39- Unload Dependencies (UD)



شکل ۴: نحوه عملکرد بُن‌سازه با سیاست انتظار در مقایسه با بُن‌سازه بدون سیاست

یک کانتینر جدید برای درخواست ورودی جدید ایجاد کند.

شـــکل (٤) نحوه عملکرد روش مدنظر سیاست انتظار را نمایش می‌دهد. همان‌طور که در این شکل مشاهده می‌شود درخواست شماره ۲ قبل از اتمام اجرای تابع در کانتینر شماره ۱ رسیده است. این درخواست نیاز به منابع پردازشی مشابهی با منابع موجود در کانتینر ۱ دارد. تشابه در این منابع می‌تواند شامل یکســـان بودن زبان مورد نیاز برای اجرای توابع۱ و ۲ و همچنین تشـــابه در وابستگی‌های این دو تابع باشد. در حالت اول، یعنی وقتی که بُن‌سازه از سیاست انتظار پیروی نمی‌کند، بلافاصله بعد از رسیدن درخواست، بُن‌سازه اقدام به ایجاد یک کانتینر جدید برای درخواست ورودی می‌کند. در حالت دوم، یعنی وقتی که بُن‌سازه از سیاست انتظار پیروی می‌کند، ابتدا بررســـی می‌کند که آیا درخواست ۲ می‌تواند از کانتینر ۱ استفاده کند یا نه. همچنین بُن‌سازه بررسی می‌کند که منتظر ماندن زمان کمتری لازم دارد یا ایجاد یک کانتینر جدید. با توجه به اینکه زمان انتظار در این مثال کمتر از زمان ایجاد کانتینر جدید و متحمل شدن شروع سرد است، بُن‌سازه تصمیم میگیرد که درخواست ۲ منتظر بماند و پس از اتمام کار تابع ۱، از منابع موجود در کانتینر ۱ استفاده کند.

٤-۱- مدل ریاضی مسئله

به منظور اجرای سیاســـت انتظار، بُن‌ســـازه نیاز دارد اطلاعـــات مختلفی از کانتینرهای درحـــال اجرا و همچنین درخواست ورودی بداند تا بتواند محاسبات لازم در هنگام

جدول ۱ :نمادهای استفاده شده در مدل ریاضی

توضیح	نماد
لیست کانتینرهای فعال	$LAC = \{c_1, ..., c_n\}$
زبان اجرای تابع i	LF_i
زمان بارگذاری زبان تابع i	TLF_i
لیست وابستگی‌های تابع i	$DF_i=\{df_1, ..., df_n\}$
زمان اجرای تابع i	$TF_i=\{t1f_i, ..., tnf_i\}$
زمان سپری‌شده از اجرای تابع i	TSF_i
زمان تحویل تابع i	TUF_i
درخواست ورودی	R_j
زبان اجرای درخواست j	LR_j
زمان بارگذاری زبان درخواست j	TLR_j
لیست وابستگی‌های درخواست j	$DR_j=\{dr1, ..., dr_n\}$
لیست وابستگی‌های مشترک	LCD
زمان بارگذاری وابستگی مشترک kام	$\lvert LCD_k \rvert$
امتیاز وابستگی‌های مشترک	$Score_{LCD}$
امتیاز زبان مشترک	$Score_L$
امتیاز تشابه	S_{score}
زمان انتظار	T_{wait}
سود انتظار	Benefit

ورود یـــک درخواســـت را انجام‌دهد. این اطلاعات شـــامل زبان اجـــرای کانتینرهای فعال، وابســـتگی‌های بارگذاری شـــده در کانتینرهای فعال، زمان اجرای توابع کانتینرهای فعال، زمان تحویل توابع کانتینرهای فعال، وابســـتگی‌های مورد نیاز درخواست ورودی و زبان مورد نیاز درخواست ورودی می‌شـــود. بـــا توجه به ایـــن اطلاعات، بُن‌ســـازه درهنگام ورود یک درخواســـت امتیاز تشابه کانتینر فعال با درخواســـت ورودی، مدت زمان انتظار و سود انتظار را محاسبه می‌کند. محاسبه این موارد به بُن‌سازه کمک می‌کند تا تصمیم درستی درباره درخواست ورودی بگیرد. جدول (۱) نمادهای استفاده شده در مدل ریاضی محاسبهٔ تشابه، مدت زمان انتظار و سود انتظار را نشان می‌دهد.

٤-۱-۱- محاسبه امتیاز تشابه

بـــرای محاســـبه امتیاز تشـــابه درخواســـت ورودی بـــا کانتینرهای فعـــال ابتدا نیـــاز داریم میزان تشـــابه در وابســـتگی‌های توابـــع در حـــال اجرا را با وابســـتگی‌های درخواست ورودی به دست آوریم ، برای این کار اشتراک لیست وابستگی‌های تابع i و لیست وابستگی‌های درخواست

Algorithm 1: Calculate Similarity Score

Input: DFi, DRj, TDFi, LFi, LRj, TLFi
Result: A numerical value indicating the similarity

```
Score ← 0;
while !isEmpty(DFi) do
    df ← DFi.pop();
    if df in DRj then
        Score ← Score + TDFi[df];
    end
end
if LFi = LRj then
    Score ← Score + TLFi;
end
return Score
```

شکل ۵: شبه‌کد امتیاز تشابه

j را با اســتفاده از فرمول (۱-٤) محاســبه می‌کنیم. سپس مدت زمان‌هــای لازم برای بارگذاری این وابســتگی‌های مشترک را با استفاده از فرمول (۲-٤) به دست می‌آوریم.در مرحله بعد اگر زبان کانتینر فعال با زبان درخواست جدید یکی باشــد امتیاز زبان با فرمول (۳-٤) محاسبه می‌شود. درنهایت امتیاز تشابه درخواست ورودی با کانتینر فعال با فرمول (٤-٤) زیر محاسبه می‌گردد:

$$LCD = DF_i \cap DR_j \quad (۱-٤)$$

$$Score_{LCD} = \sum_{k=0}^{n} |LCD_k| \quad (۲-٤)$$

$$Score_L = TLF_i \; if \; LF_i = LR_j \; else \; 0 \quad (۳-٤)$$

$$S_{score} = Score_{LCD} + Score_L \quad (٤-٤)$$

مطابق شــبه‌کد شکل (۵) ابتدا میزان امتیاز را به اندازهٔ زمان لازم برای بارگذاری هر وابســتگی مشترک، افزایش می‌دهیــم و در آخر اگر زبان کانتینر فعال و درخواســت ورودی یکــی بــود، امتیــاز را به اندازهٔ زمــان لازم برای بارگذاری زبــان افزایش می‌دهیم و بدیــن صورت امتیاز تشابه به دست می‌آید.

۴-۱-۲ محاسبه مدت زمان انتظار

مــدت زمان انتظار، زمانی اســت که درخواســت باید منتظــر بماند تــا اجرای تابع فعــال در کانتینر مشــابه با درخواست به پایان برســد. برای محاسبه این زمان لازم اســت زمان اجرای هر تابع را داشــته باشیم. بدین منظور بُن‌سازه از آغاز فعالیتش سابقهٔ زمان‌های اجرای توابع را

Algorithm 2: Calculate Waiting Time

Input: TFi, TSFi, TUFi
Result: A numerical value indicating the waiting time

```
TFi ← AVG(TFi);
Twaiting ← TFi − TSFi;
Twaiting ← Twaiting + TUFi;
return Twaiting
```

شکل ۶: شبه‌کد محاسبه مدت زمان انتظار

دخیره می‌کند و بر اســاس فرمول (۵-٤) مقدار TFi برابر با میانگین ســابقهٔ زمان‌های اجرای تابع i خواهدبود. حال، برای محاسبه مدت زمان انتظار کافی‌است طبق فرمول (٦-٤) زمان باقی‌مانده از اجرای تابع فعال را به دست‌آورده و با مقدار زمان تحویل آن تابع جمع کنیم. شــبه‌کد شکل (٦) نحو محاسبهٔ زمان انتظار را نشان می‌دهد.

$$|TF_i| = AVG(TF_i) \quad (۵-٤)$$

$$T_{wait} = (|TF_i| - TSF_i) + TUF_i \quad (٦-٤)$$

۴-۱-۳ محاسبه سود انتظار و تصمیم‌گیری

حال با توجه به پارامترهای محاسبه‌شده در دو مرحله قبل، اکنون می‌توان ســود حاصل از انتظــار را با فرمول (۷-٤) به‌دست‌آورد.

$$Benefit = S_{score} - T_{wait} \quad (۷-٤)$$

شبه‌کد شــکل (۷) الگوریتم پیدا کردن بهترین کانتینر را نشان می‌دهد، بُن‌سازه به‌ازای تمامی کانتینرهای فعال مقدار ســود را محاســبه می‌کند و کانتینری که بیشترین سود را داشته‌باشد را انتخاب می‌کند. اگر مقدار سود بیشینه بیشتر از صفر باشد یعنی زمان انتظار به‌اندازه سود محاسبه شده، کم‌تر از زمان شــروع ســرد خواهد بود و بُن‌سازه تصمیم می‌گیرد که درخواســت را منتظر کانتینــر فعال نگه دارد و پــس از اتمام کار تابع فعــال در کانتینر مورد نظر، از منابع آن کانتینر برای اجرای تابع درخواستی استفاده‌کند. در این حالت الگوریتم بالا میزان سود بیشینه و کانتینر مورد نظر را برمی‌گرداند. در حالتی که مقدار سود بیشنه کمتر از صفر باشــد، یعنی منتظرماندن بــرای کانتینرهای فعال به‌صرفه نیست و متحمل شــدن شروع سرد بهتر خواهدبود. در این حالت الگوریتم مقدار null برمی‌گرداند.

٤-۲- تحلیل پیچیدگی الگوریتم

```
Algorithm 3: Find best container
  Input: LAC, Rj
  Result: Best container if exist
best_container ← null;
best_benefit ← 0;
DRj, LRj ← Rj;
for i in LAC do
    DFi, TDFi, LFi, TLFi, TFi, TSFi, TUFi ← i;
    Sscore ← calculate_similarity_score(DFi, DRj, TDFi, LFi, LRj, TLFi);
    Twait ← calculate_waiting_time(TFi, TSFi, TUFi);
    benefit ← Sscore − Twait;
    if benefit ≥ best_benefit then
        best_benefit ← benefit;
        best_container ← i;
    end
end
if best_containerisnotnull then
    return best_container, best_benefit
end
return null;
```

شکل ۷: شبه‌کد محاسبه سود و تصمیم‌گیری

همان‌طور که دیدیم، الگوریتم ارائه‌شده از دو زیر بخش و یک بخش اصلی تشکیل شده است. بخش اصلی الگوریتم به‌ازای هر درخواست ورودی R_j اجرا شده و در هر بار اجرای این الگوریتم، یک حلقه به تعداد کانتینرهای فعال اجرا می‌شود. در هر بار اجرای این حلقه، دو زیر بخش این الگوریتم که اولی برای محاسبه امتیاز تشابه و دومی برای محاسبه مدت زمان انتظار هستند اجرا می‌شوند و در ادامه با اجرای یک شرط کانتینر مناسب برای اعمال سیاست انتظار مشخص می‌شود. در بخش محاسبه امتیاز تشابه لازم است که تشابه وابستگی‌های تابع درحال اجرا و تابع درخواستی بررسی شود. بنابراین، پیچیدگی محاسباتی این بخش از مرتبه تعداد وابستگی‌های تابع درحال اجراست که این پیچیدگی را با $O(N_1)$ نمایش می‌دهیم. همچنین پیچیدگی محاسباتی بخش محاسبه مدت زمان انتظار با توجه به این‌که روی زمان‌های اجرای یک تابع میانگین می‌گیرد از مرتبه تعداد دفعات اجرای تابع است که این پیچیدگی را با $O(N_2)$ نمایش می‌دهیم. مرتبه اجرایی بخش اصلی الگوریتم نیز به تعداد کانتینرهای فعال است که این پیچیدگی را با $O(N)$ نمایش می‌دهیم. بنابراین، مرتبه پیچیدگی محاسباتی کل الگوریتم برابر با $O(NN_1 + NN_2)$ خواهد بود.

۵ ارزیابی

برای ارزیابی روش پیشنهادی، معیارهای زمان پاسخ (زمانی که درخواست برای اجرای تابع وارد بُن‌سازه می‌شود، تا زمانی که اولین پاسخ از تابع به‌دست می‌آید) و تعداد کانتینرهای فعال (مصرف منابع) در نظر گرفته شدند. برای محاسبهٔ بهبود زمان پاسخ نیاز به معیار سود[40] داریم. سود عبارت است از مقدار زمان کاهش داده شده برای پاسخ‌دهی به یک درخواست به هنگام اعمال سیاست نسبت به حالتی که سیاستی بر بُن‌سازه حاکم نیست. علاوه بر این از آنجایی که قصد و هدف ما کاهش شروع سرد است، یکی از معیارهای ارزیابی روش پیشنهادی درصد کاهش رخداد شروع سرد خواهد بود. برای به دست‌آوردن این معیار ابتدا لازم است تعداد رخداد شروع سرد در هنگام اعمال سیاست‌های مختلف را به‌دست آوریم.

۵-۱- شبیه‌سازی

برای پیاده‌سازی روش پیشنهادی یک موتور شبیه‌سازی با استفاده از زبان پایتون[41] آماده شده است که یک بُن‌سازه را شبیه‌سازی می‌کند. این موتور با توجه به درخواست‌های ورودی و سیاست حاکم برا آن بُن‌سازه اقدام به ساخت کانتینر و مدیریت کانتینرهای

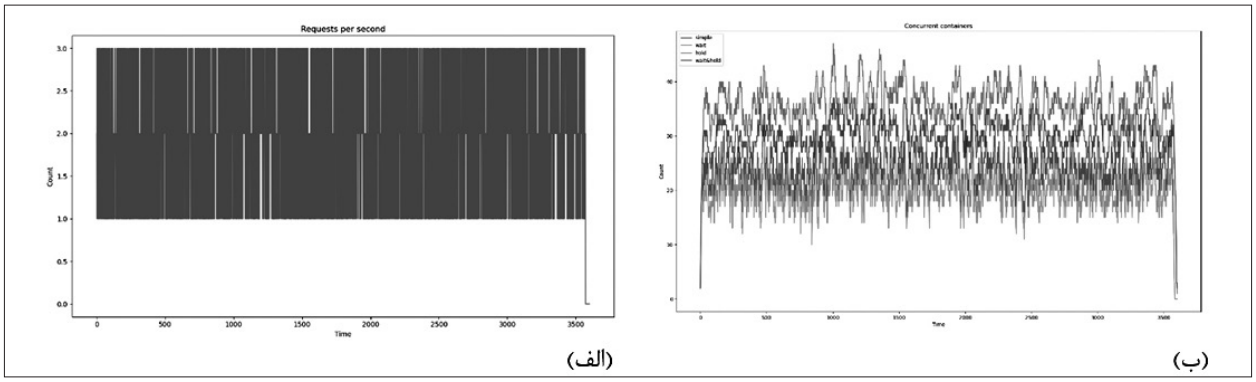
40- Benefit
41- python

موجود می‌کند و همچنین معیارهای لازم را در حین اجرای شبیه‌سازی محاسبه کرده و در پایان اجرا گزارش می‌کند. موتور قادر به اعمال چهار سیاست مختلف برای مدیریت درخواست‌های ورودی است. با سیاست بی‌تفاوت (بدون سیاست)، موتور به ازای هر درخواست ورودی یک کانتینر تخصیص می‌دهد. با سیاست انتظار، در زمان ورود درخواست، اگر کانتینر مناسبی یافت‌شد، درخواست، منتظر آن کانتینر می‌ماند. با سیاست متوقف نگه‌داشتن کانتینر، کانتینر پس از پاسخ‌دهی به یک درخواست، به اندازه یک مقدار ثابت متوقف می‌ماند. اگر در این زمان درخواست مشابهی دریافت شده دوباره برای پاسخ فعال می‌شود و اگر درخواستی دریافت نشد، کانتینر حذف می‌شود. سیاست آخر سیاست ترکیبی نام دارد که حاصل اعمال دو سیاست انتظار و متوقف نگه‌داشتن کانتینر به‌طور همزمان است. با اجرای موتور شبیه‌سازی ابتدا نیاز است که لیست درخواست‌ها در اختیار موتور قرار گیرد. این بخش، با توجه به توزیع مدنظر برای نرخ درخواست‌ها، اقدام به تولید لیست درخواست‌ها می‌کند. در این شبیه‌سازی، از پنج زبان مختلف برای ایجاد تنوع برای تولید توابع استفاده شده است. همچنین لیست وابستگی‌های هر تابع به‌صورت تصادفی شامل دو یا سه وابستگی می‌شود. همچنین، مدت زمان اجرای توابع به ازای توابع یکسان مقادیر متفاوتی دارد. پس از تولید و دریافت لیست درخواست‌ها، موتور شبیه‌سازی اقدام به اجرای حلقه اصلی شبیه‌سازی می‌کند. این حلقه به تعداد واحد زمانی که برای شبیه‌سازی در نظرگرفته شده است اجرا می‌شود و هر بار اجرای حلقه به منزله گذشت یک واحد زمانی است. در هر بار اجرا، توابع درخواستی از لیست درخواست‌ها را به بخش مدیریت کانتینرها منتقل می‌کند. قسمت مدیریت کانتینرها در موتور، وظیفه ساخت کانتینرهای جدید، به‌روزرسانی کانتینرهای موجود و اعمال سیاست بر آن‌ها را دارد. به‌روزرسانی هر کانتینر، براساس این‌که کانتینر در چه مرحله‌ای است و چه سیاستی بر موتور شبیه‌سازی حاکم است، انجام می‌شود. در مرحلهٔ بارگذاری وابستگی‌ها کانتینر به ازای هر وابستگی که لازم دارد، یک واحد زمانی صرف می‌کند. سپس بارگذاری زبان انجام می‌شود که در پیادهسازی برای آن ۲ واحد زمانی درنظر گرفته شده است. مرحلهٔ بعد بارگذاری تابع است. یک واحد زمانی صرف می‌شود تا بسته‌کد تابع بارگذاری و آماده اجرا شود. سپس تابع اجرا می‌شود و کانتینر با توجه به زمان تعیین‌شده برای اجرای تابع در مرحله ساخت لیست درخواست‌ها، چند واحد زمانی را در این مرحله می‌گذراند. پس از اجرا، به مدت یک واحد زمانی نهایی‌سازی تابع و آزادسازی[42] انجام می‌شود. در حالتی که سیاست انتظار حاکم باشد، در این مرحله بررسی می‌شود که آیا تابعی منتظر این کانتینر است یا نه. در صورت وجود، کانتینر به‌روز شده تا در مراحل بعدی بارگذاری‌های لازم برای تابع جدید را انجام دهد. در حالتی که سیاست، متوقف نگه داشتن کانتینر باشد، کانتینر پس از این مرحله به اندازه مدت زمان ثابتی که از پیش تعریف شده متوقف می‌ماند و در حالتی که سیاستی حاکم نباشد کانتینر مرحلهٔ آخر را طی می‌کند. مرحلهٔ آخر آزادسازی وابستگی‌ها و زبان و حذف کانتینر است و در آن ۳ واحد زمانی صرف می‌شود. در پایان کار موتور شبیه‌سازی، اطلاعات آماری که درهنگام اجرای شبیه‌سازی به‌دست آمده‌اند گزارش می‌شوند. این اطلاعات شامل تعداد کل درخواست‌ها، بیشینه زمان پاسخ، کمینه زمان پاسخ، میانگین زمان پاسخ، مجموع زمان پاسخ‌ها، تعداد کل کانتینرهای ساخته‌شده، مجموع تعداد کانتینرهای هم‌زمان فعال، میانگین تعداد کانتینرهای هم‌زمان فعال، مجموع سود حاصل از اعمال سیاست، میانگین سود حاصل از اعمال سیاست، تعداد کل انتظار، تعداد توقف، نمودار تعداد درخواست برحسب زمان و نمودار تعداد کانتینرهای فعال بر حسب زمان می‌شود.

۵-۲- آزمایش

برای ارزیابی روش ارائه شده دو آزمایش با مدت

42- Unload



شکل۸ : نمودار **تعداد درخواست برحسب زمان (الف) و تعداد کانتینرهای فعال برحسب زمان در آزمایش با نرخ ثابت (ب)**

اجـرای این آزمایـش نیز ۳۶۰۰ واحد زمانـی (معادل یک ساعت) طراحی شده است که در این بخش شرایط آن‌ها و اطلاعات به دست‌آمده از آن‌ها ارائه می‌شود.

۵-۲-۱- آزمایـش اول (در نظر گرفتن نـرخ ثابت برای ورود درخواست‌ها)

در ایـن آزمایـش، درخواسـت‌ها با نرخ ثابـت وارد می‌شوند. همان‌طور که در شکل (۸-الف) مشاهده می‌شود تعداد درخواست‌های ورودی در یک واحد زمانی برابر ۱، ۲ و یا ۳ اسـت. به همین دلیـل از واژه نرخ ثابت برای این آزمایش اسـتفاده شده اسـت. نمودار شکل (۸-ب) بیانگر تعداد کانتینرهای فعال در هر لحظه از زمان شبیه‌سـازی با توجه به سیاسـت اعمال شده است. رنگ نارنجی برای سیاست انتظار، رنگ سبز برای سیاست توقف، رنگ قرمز برای سیاست ترکیبی و رنگ آبی برای سیاست بی تفاوت به کار رفته‌اند.

گـزارش اطلاعات به دسـت آمده از ایـن آزمایش در جدول (۲) نشـان داده شـده اسـت. با توجه به این نتایج، معیارهای ارزیابی در جدول (۳) محاسبه شده است.

۲-۲-۵ آزمایـش دوم (در نظر گرفتن نـرخ متغیر برای ورود درخواست‌ها)

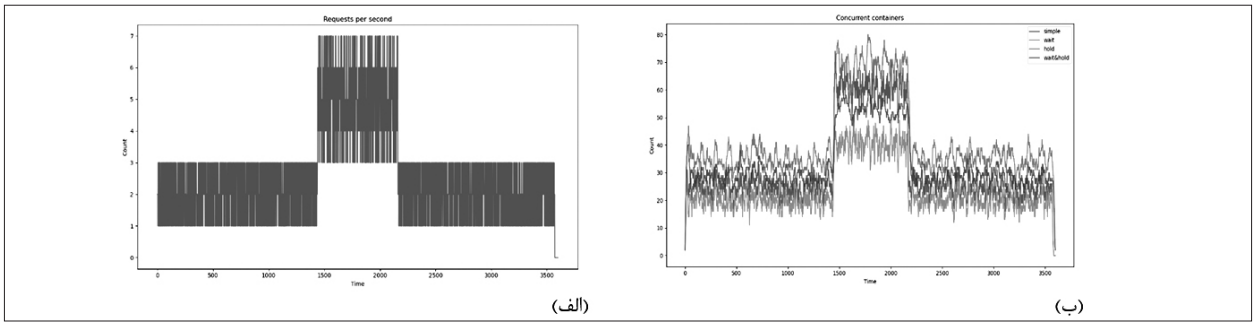
در این آزمایش نرخ دریافت درخواسـت‌ها ثابت نبوده و دارای یک اوج در ترافیک ورودی است. همان‌طور که در شکل (۹-الف) مشاهده می‌شـود، درخواست‌های ورودی به سه بخش تقسیم می‌شـود. در بخش اول و سوم تعداد درخواسـت های ورودی در یک واحـد زمانی برابر ۱، ۲ ویا ۳ اسـت. در بخش دوم تعداد درخواست‌های ورودی افزایش داشـته و و تعداد درخواست‌ها مقادیری بین ۳ تا ۷ هسـتند. مقادیر آماری به‌دسـت‌آمده نمودار شکل (۹-ب) بیانگر تعـداد کانتینرهای فعال در هـر لحظه از زمان شبیه‌سازی با توجه به سیاست اعمال شده است.

جدول۲ : مقادیر ثبت‌شده در آزمایش نرخ ثابت

کمیت آماری	مقدار ثبت‌شده باتوجه به سیاست حاکم			
	بدون سیاست	انتظار	توقف کانتینر	ترکیبی
تعداد کل درخواست	۷۰۳۲			
بیشینه زمان پاسخ	۱۰	۱۰	۱۰	۱۰
کمینه زمان پاسخ	۶	۳	۲	۲
میانگین زمان پاسخ	۸.۲۳۳۲	۷.۵۳۰۸	۵.۹۷۳۲	۵.۸۴۸۹
مجموع زمان پاسخ‌ها	۵۷۸۹۶	۵۲۹۵۷	۴۲۰۰۴	۴۱۱۳۰
تعداد کل کانتینرها ساخته‌شده	۷۰۳۲	۴۷۱۰	۱۶۱۲	۱۰۸۳
مجموع تعداد کانتینرهای هم‌زمان فعال	۸۶۰۲۴	۷۰۲۰۷	۱۲۹۵۰۲	۱۰۸۹۶۷
میانگین تعداد کانتینرهای همزمان فعال	۲۳.۸۹	۱۹.۵	۳۵.۹۷	۳۰.۲۶
مجموع سود حاصل از اعمال سیاسیت	۰	۴۹۳۹	۱۵۸۹۲	۱۶۷۶۶
میانگین سود حاصل از اعمال سیاست	۰	۰.۷۰۲۳	۲.۲۵۹۹	۲.۳۸۴۲
تعداد کل انتظار	۰	۲۳۲۲	۰	۱۱۵۰
تعداد کل توقف	۰	۰	۵۴۲۰	۴۷۹۸

جدول۳ : ارزیابی سیاست‌های مختلف در آزمایش نرخ ثابت

معیار ارزیابی	سیاست انتظار	سیاست توقف	سیاست ترکیبی
بهبود زمان پاسخ	٪۸,۵	٪۲۷,۴	٪۲۸,۹
کاهش رخداد شروع سرد	٪۳۳	٪۷۷	٪۸۴,۵
بهینه‌گی مصرف منابع	+ ٪۱۳,۳	- ٪۳۶,۷	- ٪۱۹,۳



شکل ۹ : نمودار تعداد درخواست برحسب زمان (الف) و تعداد کانتینرهای فعال برحسب زمان در آزمایش با نرخ غیر ثابت (ب)

جدول۴ : مقادیر ثبت‌شده در آزمایش نرخ متغیر

کمیت آماری	مقدار ثبت‌شده باتوجه به سیاست حاکم			
	بدون سیاست	انتظار	توقف کانتینر	ترکیبی
تعداد کل درخواست	۹۴۰۱			
بیشینه زمان پاسخ	۱۰	۱۰	۱۰	۱۰
کمینه زمان پاسخ	۶	۳	۲	۲
میانگین زمان پاسخ	۸.۱۹۵۹	۷.۱۹۴۶	۵.۶۴۵۳	۵.۵۴۳۱
میانگین زمان پاسخ در زمان اوج	۸,۱۸۲۵	۶.۸۶۷۷	۵.۴۵۶۷	۵,۳۲۰۳
مجموع زمان پاسخ‌ها	۷۷۰۵۰	۶۷۶۳۷	۵۳۰۷۲	۵۲۱۱۱
تعداد کل کانتینرها ساخته‌شده	۹۴۰۱	۵۱۱۱	۱۴۸۱	۸۷۳
تعداد کانتینرهای ساخته شده در زمان اوج	۳۵۹۷	۱۵۱۱	۳۸۳	۱۶۶
مجموع تعداد کانتینرهای هم‌زمان فعال	۱۱۴۶۵۷	۸۵۰۹۰	۱۵۱۵۸۶	۱۲۲۰۵۰
میانگین تعداد کانتینرهای هم‌زمان فعال	۳۱,۸۴	۲۳,۶۳	۴۲,۱۰	۳۳,۹۰
مجموع تعداد کانتینرهای هم‌زمان فعال در زمان اوج	۴۳۵۵۰	۲۹۰۰۷	۴۹۷۱۳	۳۷۷۸۸
میانگین تعداد کانتینرهای هم‌زمان فعال در زمان اوج	۶۰,۴۸	۴۰,۲۸	۶۹,۰۴	۵۲,۴۸
مجموع سود حاصل از اعمال سیاسیت	۰	۹۴۱۳	۲۳۹۷۸	۲۴۹۳۹
میانگین سود حاصل از اعمال سیاست	۰	۱,۰۰۱۲	۲,۵۵۰۵	۲,۶۵۲۸
میانگین سود حاصل از اعمال سیاست در زمان اوج	۰	۱.۳۴۷۹	۲.۷۲۷۱	۲.۶۳۹۳
تعداد کل انتظار	۰	۴۲۹۰	۰	۱۹۰۰
تعداد کل توقف	۰	۰	۷۹۱۹	۶۶۲۷

گـــزارش اطلاعات به دســـت آمده از ایـــن آزمایش در جدول (٤) نشـــان داده شـــده اســـت. با توجه به این نتایج، معیارهای ارزیابی در جدول (٥) محاسبه شده است.

۶- نتیجه‌گیری

در این پژوهش در پی یافتن روشی برای بهبود مشکل تاخیر شروع سرد در رایانش بی‌خدمت‌گذار بودیم. تاخیر شـــروع ســـرد از مهم‌ترین چالش‌هایی اســـت کـــه قابلیت مقیاس‌پذیـــری خودکار و مقیاس‌پذیری به صفر را با خود به همراه دارد و باعث کاهش معیارهای کیفی اســـتفاده از

جدول ۵ : ارزیابی سیاست‌های مختلف در آزمایش نرخ متغیر

معیار ارزیابی	سیاست انتظار	سیاست توقف	سیاست ترکیبی
ارزیابی سیاست‌های مختلف در آزمایش نرخ متغیر			
بهبود زمان پاسخ	٪۱۲,۲	٪۳۱,۱	٪۳۲,۳
کاهش رخداد شروع سرد	٪۴۵,۶	٪۸۴,۲	٪۹۰,۷
بهینه‌گی مصرف منابع	+٪۲۵,۷	-٪۳۲,۲	-٪۶,۴
ارزیابی سیاست‌های مختلف در آزمایش نرخ متغیر در زمان اوج			
بهبود زمان پاسخ	٪۱۶	٪۳۳,۳	٪۳۲,۲
کاهش رخداد شروع سرد	٪۵۷,۹	٪۸۹,۳	٪۹۵,۳
بهینه‌گی مصرف منابع	+ ٪۳۳,۳	- ٪۱۴,۱	+ ٪۱۳,۲

این مدل رایانشی می‌شود. بدین منظور ابتدا مفاهیم مورد نیاز برای درک مســـئله را مطرح کرده و سپس چند نمونه

از روش‌هایی که برای کاهش تأثیر این مشکل بر کیفیت خدمات ارائه شده‌اند را بررسی کردیم. در ادامه، روشی نو برای این مسئله مطرح کردیم که روش ارائه‌شده مبتنی بر استفاده از محیط‌های اجرایی گرم است. در این روش با بررسی کانتینرهای موجود، به دنبال کانتینرهای فعالی هستیم که منتظر ماندن برای خاتمه اجرای آن‌ها به صرفه است. در این روش، یافتن کانتینری که بتواند درخواست ما را اجرا کند و زمان پاسخ به درخواست را کاهش‌دهد، نه تنها منجر به بهبود زمان پاسخ به درخواست می‌شود و از افت کیفت خدمت ارائه شده توسط بُن‌سازه نمی‌کاهد، بلکه باعث صرفه‌جویی در مصرف منابع پردازشی بُن‌سازه نیز می‌شود و نتایج حاصل از آزمایش‌های انجام شده نیز اثباتی بر این مدعا هستند. روش پیشنهادی این پژوهش، برای بهبود و نزدیکی هرچه بیشتر به واقعیت همچنان جای مطالعه و بررسی دارد. بدین منظور پیشنهادهایی برای ادامه راه این پژوهش می‌توان ارائه‌کرد.

یکی از کارهایی که پیشنهاد می‌شود، پیاده‌سازی و بررسی این روش در یک بُن‌سازه واقعی است. همچنین آزمایش‌های مختلفی باید طراحی شود تا عملکر این روش در شرایط مختلف به خوبی بررسی شود. این کار با هدف نزدیکی هرچه بیشتر به واقعیت باید صورت گیرد تا نتایج به دست آمده از نحوه عملکرد روش ارائه‌شده دقیق‌تر و قابل استنادتر باشند. برای این کار می‌توان روش ارائه‌شده را در یکی از بُن‌سازه‌های متن باز مانند بُن‌سازه OpenWhisk پیاده‌سازی کرد. از آنجایی که بُن‌سازه OpenWhisk نیز از روش متوقف نگه‌داشتن کانتینرها برای کاهش شروع سرد استفاده می‌کند، بُن‌سازه مناسب‌تری به نظر می‌رسد و می‌توان نتایج به دست آمده از پیاده‌سازی واقعی را با نتایج شبیه‌سازی این پژوهش مقایسه نمود.

در بخش پیاده‌سازی و ارزیابی، روش ارائه‌شده را با روش توقف کانتینر مقایسه کرده و با این روش ترکیب کردیم. می‌توان روش ارائه‌شده را با دیگر روش‌هایی که تاکنون ارائه شده‌اند مقایسه و ترکیب کرد. از آنجایی که ترکیب این روش با روش سادهٔ توقف کانتینر نتیجهٔ خوبی داشت، پیش‌بینی می‌شود که ترکیب این روش با روش‌های بهتر و پیچیده‌تری که تا کنون ارائه شده‌اند نتایج خوبی در بر داشته باشد. برای مثال با ترکیب کردن این روش با روشی که بُن‌سازه آزور[43] به‌کار می‌برد می‌توان نتایج بهتری به دست‌آورد. این بُن‌سازه نسبت به روش متوقف نگه‌داشتن کانتینرها هوشمندانه‌تر عمل‌می‌کند و باتوجه به تاریخچه فراخوانی‌ها مدت زمان متفاوتی را برای گرم نگه‌داشتن کانتینرها، در نظر می‌گیرد. باتوجه به این موضوع، پیش‌بینی می‌شود که ترکیب روش ارائه‌شده در این پژوهش با این روش، در مصرف منابع صرفه‌جویی بیشتری داشته‌باشد.

برای ارزیابی روش ارائه شده فقط ۳ معیار ارزیابی ارائه و بررسی شدند، می‌توان از معیار مختلف دیگری نیز برای ارزیابی استفاده نمود. برای مثال می‌توان میزان سربار پردازش، سربار ورود و خروج[44] و سربار حافظه را با اجرای این روش محاسبه نموده و با روش های مختلف مقایسه کرد.

مراجع

[1] S. Mohit and S. Sachchidanand, “Winning in the Era of Serverless Computing and Function as a Service,” in Proceedings of the 2018 3rd International Conference for Convergence in Technology (I2CT), 6-8 April, Delhi, India, pp. 1-5, 2018.

[2] B. Ioana, C. Paul, C. Kerry, C. Perry, F. Stephen, I. atche, M. Nick, “Serverless computing: Current trends and open problems.,” Research Advances in Cloud Computing, Singapore, vol.12, no.5, 2017.

[3] S.A. Bello, L.O. Oyedele, O.O. Akinade, M. Bilal, J.M. Delgado, LA. Akanbi, A.O. Ajayi, H.A. Owolabi, “Cloud computing in construction industry: Use cases, benefits and challenges”, Automation in Construction, vol. 122, 2021.

[4] U. F. Minhas, “A Performance Evaluation of Database Systems on Virtual Machines,” Concurrency and Computation: Practice and Experience, vol. 32, no. 7, 2007.

[5] B. Bashari Rad, H. John Bhatti, M. Ahmadi, “An Introduction to Docker and Analysis of its Performance,” IJCSNS International Journal of Computer Science and Network Security, vol.17 no.3, 2017.

[6] R. Perrey, M. Lycett, “Service-oriented Architecture”, in Proceedings of the 2003 Symposium on Applications and the

43- Azure
44- Input Output

IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), June 5- 8, Atlanta, USA, 2017, pp.114-119.
[15] A. Istemi Ekin, C. Ruichuan, R. Ivica, S. Manuel, S. Klaus, B. Andre, A. Paarijaat, and H. Volker, "SAND: Towards High-Performance Serverless Computing," in Proceedings of the USENIX Annual Technical Conference, July 11–13, Boston, USA, 2018, pp.78-83.
[16] P. Vahidinia, B. Farahani and F. S. Aliee, "Mitigating Cold Start Problem in Serverless Computing: A Reinforcement Learning Approach," IEEE Internet of Things Journal, doi: 10.1109/JIOT.2022.3165127.
[17] D. Khan, B. Subba, S. Sharma, "Minimizing Cold Start Times in Serverless Deployments", in Proceedings of the 2022 14th International Conference on Contemporary Computing, August 5-7, Pages 156–16.
[18] P. Silva, D. Fireman, T. Emmanuel Pereira, in Proceedings of the Middleware '20: 21st International Middleware Conference, December 2020, Pages 1–13.
[19] A.P. Jegannathan, R. Saha, S.K. Addya, "A Time Series Forecasting Approach to Minimize Cold Start Time in Cloud-Serverless Platform," in Proceedings of the IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), June 6-9, Sofia, Bulgaria, 2022, pp.34-41.
[20] B. Sethi, S.K. Addya, S.K. Ghosh, "LCS : Alleviating Total Cold Start Latency in Serverless Applications with LRU Warm Container Approach," in Proceedings of the 24th International Conference on Distributed Computing and Networking, Jan 4-7, Kharagpur, India, 2023 pp.77-86.
Internet Workshops, 27-31 January,Orlando, FL, USA, 2003.
[7]M.H. Valipour, B. Amirzafari, K. Niki Maleki, N.Daneshpour, "A brief survey of software architecture concepts and service oriented architecture", in Proceedings of the 2009 2nd IEEE International Conference on Computer Science and Information Technology, 08-11 August 2009, Beijing, China, pp. 34-38.
[8] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," International Journal of Cooperative Information Systems, vol. 17, no. 2, pp. 223-255, 2008.
[9] I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen, Microservice architecture: Aligning principles, practices, and culture, O'Reilly Media Inc, 2016.
[10] O. Edward, Y. Leon, Zh. Dennis, H. Kevin, H. Tyler, A. Andrea, and A. Remzi, "{SOCK}: Rapid Task Provisioning with Serverless-Optimized Containers," in Proceedings of the USENIX Annual Technical Conference ({USENIX}{ATC} 18), July 11–13, Boston, USA, 2018, pp.112-118.
[11] S. Simon, "A provider-friendly serverless framework for latency-critical applications," in Eurosys Doctoral Workshop, April, 4-8, Porto, Portugal , 2018.
[12] L. Ping-Min, and G. Alex, "Mitigating Cold Starts in Serverless Platforms: A Pool-Based Approach," arXiv Preprint, 2019.
[13] M. Shilkov, "Comparison of Cold Starts in Serverless Functions across AWS, Azure, and GCP," 5 January 2021. [Online]. Available: https://mikhail.io/serverless/coldstarts/big3/.
[14] M. Garrett, and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in Proceedings of the