



دانشگاه شهید بهشتی

دانشکده مهندسی و علوم کامپیوتر

کاهش شروع‌های سرد در پلتفرم‌های بدون سرور

گزارش سمینار کارشناسی ارشد مهندسی کامپیوتر
گرایش نرم‌افزار

نگارش

امیرمحمد کرمزاده

استاد راهنما

دکتر علیرضا شاملی

خرداد ۱۴۰۰

چکیده

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ و با استفاده از طراحان گرافیک است. چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است و برای شرایط فعلی تکنولوژی مورد نیاز و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد. کتابهای زیادی در شصت و سه درصد گذشته، حال و آینده شناخت فراوان جامعه و متخصصان را می طلبد تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی و فرهنگ پیشرو در زبان فارسی ایجاد کرد. در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

واژگان کلیدی: رایانش ابری، رایانش بدون سرور، شروع‌های سرد، FaaS، function-as-a-service

فهرست مطالب

۱	مقدمه	۱
۲	۱.۱ صورت مسئله	۲
۲	۲.۱ انگیزه‌ی تحقیق	۲
۳	۳.۱ مثال مرتبط	۳
۴	۴.۱ اهمیت موضوع	۴
۵	۵.۱ نتیجه‌های مهم تحقیق	۵
۶	۲ مروری بر ادبیات	۶
۷	۱.۲ رایانش بدون سرور	۷
۷	۱.۱.۲ تعریف رایانش بدون سرور	۷
۱۰	۲.۱.۲ معماری	۱۰
۱۱	۳.۱.۲ ویژگی‌های پلتفرم‌های بدون سرور	۱۱
۱۴	۴.۱.۲ پلتفرم‌های تجاری	۱۴
۱۷	۵.۱.۲ پلتفرم‌های آزاد و متن باز	۱۷
۱۷	۶.۱.۲ Function-as-a-Service	۱۷
۱۸	۲.۲ Scale-to-Zero	۱۸
۱۹	۳.۲ تاخیر شروع سرد	۱۹

۲۱

۳ کارهای مرتبط

۲۳

۴ نتیجه‌گیری

۲۵

مراجع

فهرست شکل‌ها

۴	اهمیت شروع سرد	۱.۱
۹	مرزهای رایانش بدون سرور و رایانش سرورآگاهانه	۱.۲
۱۰	معماری کلی یک پلتفرم بدون سرور	۲.۲
۱۹	اهمیت زبان و پلتفرم بدون سرور در تاخیر شروع سرد	۳.۲
۲۰	اهمیت اندازه کتابخانه برنامه در تاخیر شروع سرد	۴.۲

فهرست جداول

فصل ۱

مقدمه

۱.۱ صورت مسئله

چه روش‌هایی برای کاهش تعداد شروع‌های سرد در پلتفرم‌های بدون سرور^۱ با حداقل سربرار^۲ و زمان اجرایی وجود دارند؟

۲.۱ انگیزه‌ی تحقیق

رایانش بدون سرور^۳ یکی از مسائل داغ و محبوب این‌روزهای دنیای مهندسی نرم‌افزار و رایانش ابری است. رایانش بدون سرور حوزه‌ی جدیدی را در توسعه‌ی محصول و استقرار^۴ اپلیکیشن‌ها باز کرده است. یکی از دلایل محبوبیت استفاده از پلتفرم‌های بدون سرور و تمایل توسعه‌دهندگان برای مهاجرت به سمت آن، استفاده بیش از پیش از معماری میکروسرویس و نانوسرویس در توسعه‌ی محصولات و حرکت معماران و مهندسين نرم‌افزار در تولید و مهاجرت برنامه‌های کاربردی با این معماری‌ها است.

از دید توسعه دهنده، رایانش ابری با حذف دخالت مستقیم کاربران انتهای در مدیریت زیرساخت از جمله IaaS یا IaaS-like یا IaaS-like موجب بهبود سرعت توسعه محصول و تمرکز کاربران بر روی منطق^۵ برنامه است. همچنین، برای علاوه بر آسانی استفاده و پنهان‌سازی پیچیدگی مدیریت سرور از کاربر، به علت اینکه ارائه‌دهندگان خدمات ابری در نقاط مختلف جهان حضور دارند و همچنین کانفیگ بهینه CDN ها؛ ارتباطات بین سرورها و کاربران با حداقل تاخیر^۶ صورت می‌گیرد.

به طور کلی، یک پلتفرم بدون سرور را هر پلتفرم محاسباتی تعریف کرد که در آن مدیریت مستقیم سرور از کاربران مخفی شده و برنامه‌های کاربردی به صورت اتوماتیک در آن مقیاس‌پذیر می‌شوند و تنها هنگامی که در حال استفاده از پلتفرم هستیم، هزینه آن را پرداخت می‌کنیم. [۱]

یکی از قابلیت‌هایی که در رایانش بدون سرور باعث محبوبیت آن شده است، قابلیت Scale-to-Zero است. این بدان معنی است که هنگامی که از یک کانتینر استفاده‌ای نداریم، منابع آن گرفته می‌شوند و کانتینر اصطلاحاً

¹serverless

²Overhead

³Serverless Computing

⁴Deployment

⁵logic

⁶Latency

Zero-Scaled می‌شود. این خود موجب قابلیت پرداخت تنها در حین مصرف ما از تابع می‌شود. اما مشکل اصلی زمانی است که درخواست جدیدی برای کانینر Zero-Scale شده می‌رسد؛ در این حالت باید درخواست منتظر مانده تا سلسله‌ای از آماده‌سازی‌ها انجام شوند تا کانینر مربوطه مجدداً اجرا شود. این خود باعث تاخیری مضاف برای پاسخ‌دهی به درخواست را موجب می‌شود که به این تاخیر مشکل شروع سرد^۱ گفته می‌شود. در واقع می‌توان گفت تاخیر شروع سرد ناشی از تلاش ما در تعادل بین تاخیر در پاسخگویی به درخواست‌ها و هزینه (هزینه‌های استفاده از رم و سی‌پی‌یو و ...) است.

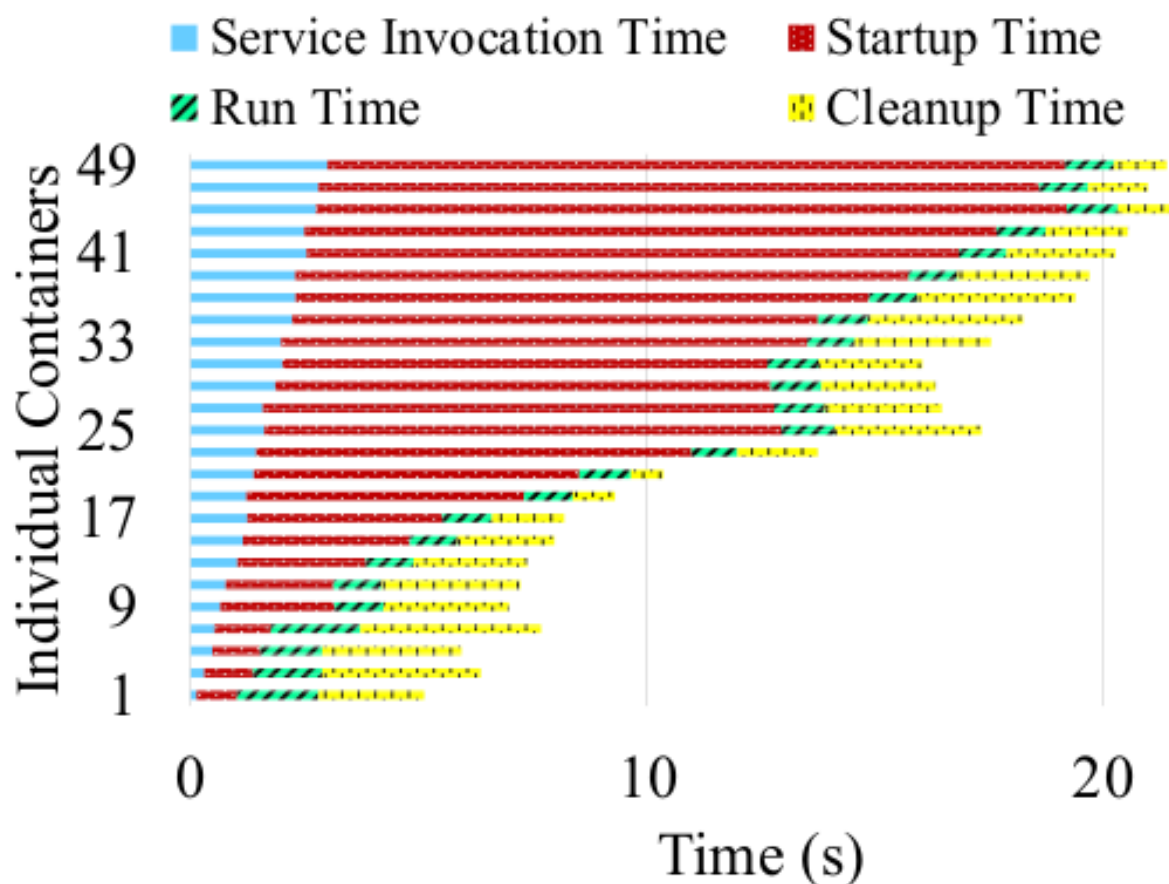
متأسفانه در سالیان اخیر و در عین داغ‌بودن مبحث و نیاز بازار به حل این مشکل، این مشکل چندان در محیط‌های آکادمیک مورد بررسی قرار نگرفته است. البته در نگاه کلی‌تر، مشکلات و مسائل بار مربوط به رایانش بدون سرور، اکثراً در محیط‌های آکادمیکی مثل دانشگاه‌ها با کم محلی روبرو شده‌اند. در این میان، پلتفرم‌های متن‌باز^۲ که دارای جوامع بسیار گسترده‌ای نیز هستند، به خاطر این سری مسائل باز که شرکت‌های تجاری در حال صرف هزینه‌های هنگفتی برای حل و فصل مشکلات مربوط به آن هستند، به شدت از رقابت عقب مانده‌اند. انگیزه ما برای انجام این پژوهش در این است که اولاً بتوانیم به راهکار مناسب‌تری برای حل مشکل مربوط به شروع سرد در پلتفرم‌های بدون سرور برسیم و ثانیاً بتوانیم با مشارکت در بهبود یکی از پلتفرم‌های آزاد در توسعه این پلتفرم‌ها تاثیر کوچکی داشته باشیم.

۳.۱ مثال مرتبط

در مقاله [۲] آقای لین و همکارش توانستند تا با اسفاده از استخر گرم و نگره‌داری کانینر توابعی که محبوبیت استفاده دارند، مدت زمان پاسخ را در حدود ۸۵٪ کاهش دهند. این بهبود در پلتفرم knative اجرا شد و ایده‌ی مقالات دیگری نیز بوده است.

^۱Cold Start

^۲Open Source



شکل ۱.۱: اهمیت شروع سرد

۴.۱ اهمیت موضوع

اگرچه پلتفرم‌های بدون سرور از نظر هزینه، scaling، راحتی استفاده و گستردگی پوشش جغرافیایی برای ما بهینه هستند؛ اما مشکل شروع سرد مشکلی نیست که بتوان به سادگی از آن گذر کرد. تصویر ۱.۱ که از [۳] گرفته شده است، نشان می‌دهد که بیش از ۸۰٪ زمان اجرای کامل یک کانتینر در پلتفرم‌های بدون سرور به آماده‌سازی آن یا شروع سرد اولیه^۱ مربوط می‌شود.

ستون قرمز رنگ زمان آماده‌سازی کانتینر را نمایش می‌دهد. این زمان همان زمان شروع سرد است که دلایل مختلفی از جمله آماده‌سازی کانتینر یا حضور در صف انتظار برای تخصیص منابع باشد. بنابراین، با به‌کارگیری یک استراتژی مناسب می‌توان این زمان را به حداقل رسانید.

متأسفانه تاخیر شروع سرد باعث شده تا توسعه‌دهندگان اقبال کمتری به استفاده از پلتفرم‌های بدون سرور

¹First Cold Start

داشته باشند. به گونه‌ای که از بین ۱۰۰۰ برنامه‌ی کاربردی بزرگ در پلتفرم Microsoft Azure، تنها یک مورد مربوط به یک برنامه تجاری باشد [۴]. این موضوع نشان می‌دهد که علی‌رغم پتانسیل بالای رایانش بدون سرور، وجود مشکلات جدی از جمله شروع سرد، باعث امتناع توسعه‌دهندگان از مهاجرت به پلتفرم‌های بدون سرور باشد.

۵.۱ نتیجه‌های مهم تحقیق

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می‌باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می‌طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه‌ای علی‌الخصوص طراحان خلاق، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می‌توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساساً مورد استفاده قرار گیرد.

ساختار این گزارش به این ترتیب خواهد بود:

درادبیات موضوع مروری بر واژگان، مفاهیم تخصصی و هر آنچه که در ادامه به آن نیاز پیدا خواهیم کرد، خواهیم داشت. سپس در فصل کارهای مرتبط به بیان مشروح تحقیقات انجام شده خواهیم پرداخت و در نتیجه‌گیری و کارهای آینده خلاصه‌ای از مطالعات انجام شده و مسائل باز خواهیم پرداخت.

فصل ۲

مروری بر ادبیات

در این بخش سعی داریم تا با مروری بر اصطلاحات و ابزارهای مورد استفاده در پژوهش‌های بررسی شده، با پیش‌نیازهای مبحث موردنظر آشنا شویم.

۱.۲ رایانش بدون سرور

رایانش بدون سرور^۱ در سال ۲۰۱۴ توسط شرکت آمازون برای اولین بار معرفی شد. تا قبل از این رایانش بدون سرور یک مفهوم انتزاعی^۲ در شبکه بود که شرکت آمازون با ارائه پلتفرم AWS Lambda Functions [۴] به معرفی آن پرداخت. سپس در سال ۲۰۱۶ سایر ارائه‌دهندگان خدمات ابری نیز به ارائه پلتفرم‌های بدون سرور خود پرداختند. در این سال به ترتیب شرکت‌های گوگل پلتفرم google cloud functions یا به اختصار GCP، شرکت مایکروسافت پلتفرم Microsoft Azure functions و شرکت IBM به معرفی IBM OpenWhisk پرداختند. البته باید توجه داشت که مفهوم رایانش بدون سرور به طور کامل توسط ارائه‌دهندگان خدمات ابری پیاده‌سازی نشده است و جای کار بسیاری دارد (با مطالعه این گزارش به مرور متوجه نواقص موجود خواهید شد).

در رایانش بدون سرور ما از نقطه قوت ماشین‌های مجازی که ایزولاسیون برنامه‌های مختلف از همدیگر بود استفاده کرده‌ایم. منتها این مورد را با مفهوم کانتینرها پیاده‌ کرده ایم. در ادامه راجع به کانتینرها نیز بحث خواهیم کرد.

۱.۱.۲ تعریف رایانش بدون سرور

رایانش بدون سرور مبحثی از رایانش ابری است که در آن بحث مدیریت حافظه یا Storage، مدیریت زیرساخت و بحث‌های networking با انتزاع بالایی به مصرف‌کننده می‌رسد. به عبارت دیگر، تمامی مدیریت‌های بخش‌ها بر عهده ارائه‌دهندگان است و ما اصلاً با این بحث سروکاری نداریم. در واقع، هدف اصلی رایانش بدون سرور هم این است که این پیچیدگی‌ها را از کاربر بگیرد.

به طور کلی، یک پلتفرم بدون سرور را هر پلتفرم محاسباتی تعریف کرد که در آن مدیریت مستقیم سرور از کاربران مخفی شده و برنامه‌های کاربردی به صورت اتوماتیک در آن مقیاس‌پذیر می‌شوند و تنها هنگامی که در

^۱Serverless Computing

^۲abstract

حال استفاده از پلتفرم هستیم، هزینه آن را پرداخت می‌کنیم. [۱]

بسیاری از افراد، serverless و faas را معادل یک‌دیگر می‌دانند درحالی‌که اصلا این‌گونه نیست. در ادامه راجع به این بحث به طور مفصلی بحث خواهیم کرد اما باید بدانیم که این دو مقوله کاملا جدا از همدگر هستند و مجددا تاکید می‌کنیم که رایانش بدون سرور یک مدل اجرایی در رایانش ابری است.

ازطرفی رایانش بدون سرور را باید نقطه مقابل رایانش سرور آگاهانه^۱ دانست که در آن از اطلاعات سرور در اختیار گرفته کاملا آگاهیم، کاملا بر مدیریت آن اشراف داریم و هرگونه تغییر از جمله متعادل‌سازی بارها، auto-scaling و ... باید توسط کاربر انجام شود.

یک مثال از پیاده‌سازی رایانش سرور آگاهانه را در زیرساخت به عنوان سرویس^۲ یا به اختصار IaaS است. در نقطه مقابل در رایانش بدون سرور هیچ کنترلی بر روی سرور نداریم، تنها می‌توانیم یک برنامه را بر روی سرور اجرا کنیم یا اجرای آن را به حالت تعلیق درآورده یا آن را از روی سرور حذف کنیم که هیچ‌کدام از این موارد نیز به صورت مستقیم انجام نمی‌گیرد؛ بلکه رابط گرافیکی و API وجود دارد که از طریق آن‌ها این تغییرات را اعمال می‌کنیم. بنابراین در رایانش بدون سرور، عملا هیچ راهی برای مدیریت مستقیم سرور و زیرساخت نداریم.

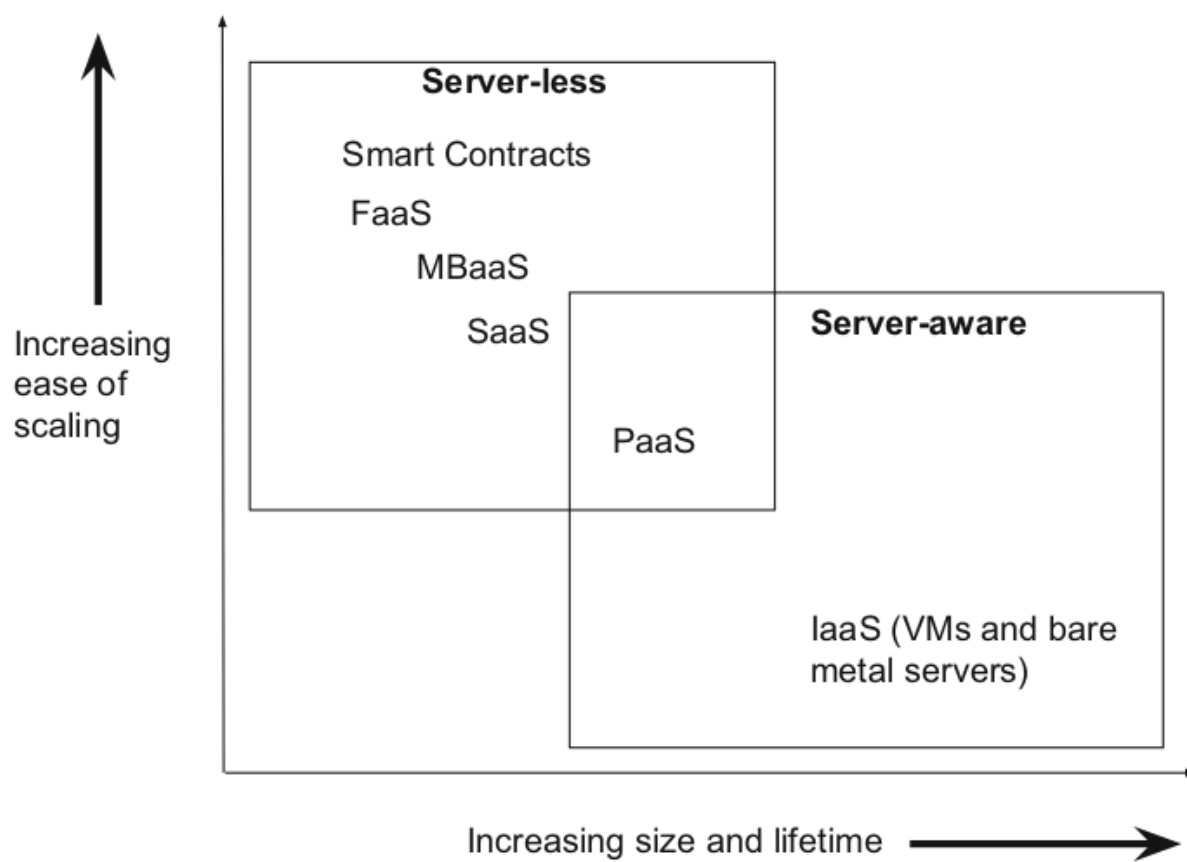
شکل ۱.۲ مرزهای بین رایانش بدون سرور و رایانش سرور آگاهانه را نمایش می‌دهد.

البته باید به این نکته توجه داشت که امروزه مرزهای بین رایانش سرور آگاهانه با رایانش بدون سرور در حال کمرنگ شدن و بعضا از بین رفتن است و این تقسیم بندی ابداء قاطعیت ندارد. همچنین تفکیک برخی موارد مانند Platform-as-a-Service یا به اختصار PaaS به راحتی انجام نمی‌گیرد بلکه این نوع رایانش می‌تواند از نوع باسرور یا بدون سرور باشد. در این شکل هرچه به سمت محور افقی حرکت می‌کنیم دانه‌بندی و طول عمر افزایش پیدا می‌کند و هرچه به سمت بالاتر می‌رویم، scaling راحت‌تر انجام می‌گیرد.

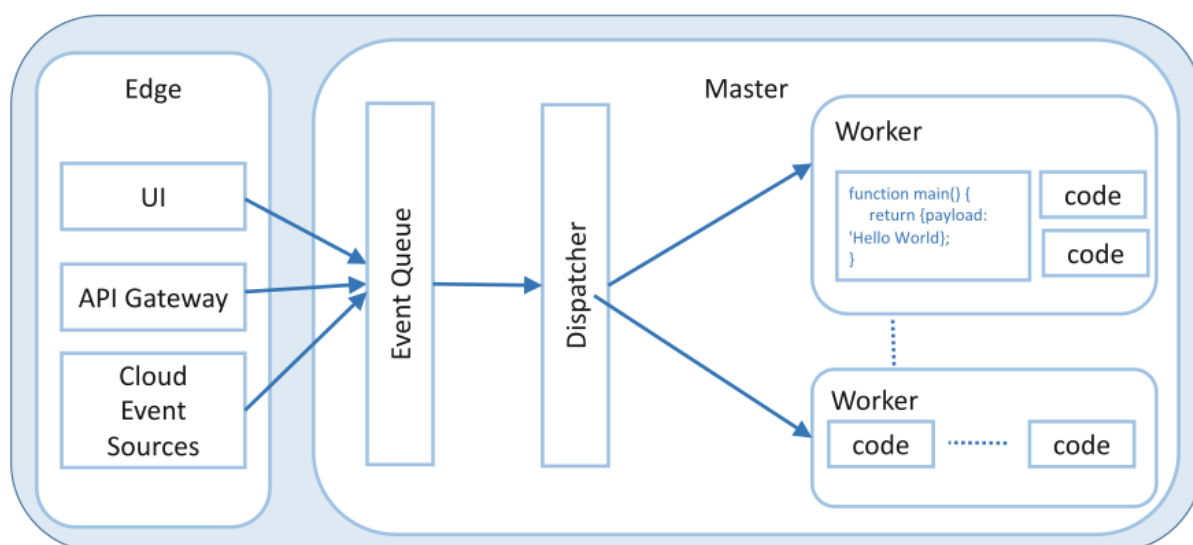
از مزایای رایانش بدون سرور همچنین می‌توان به پشتیبانی و توسعه راحت‌تر اپلیکیشن‌ها با معماری میکروسرویس و نانوسرویس هم اشاره کرد. البته معماری نانوسرویس مبحث جدیدتری است و جای پژوهش‌های بیشتری دارد.

¹ Server Aware

² Infrastructure as a service



شکل ۱.۲: مرزهای رایانش بدون سرور و رایانش سرورآگاهانه



شکل ۲.۲: معماری کلی یک پلتفرم بدون سرور

۱

۲.۱.۲ معماری

واژه serverless ممکن است این تفکر را به ذهن مبتدر سازد که اصلاً در این نوع مدل رایانشی سروری نداریم؛ در حالی که این امر بسیار اشتباه است. در رایانش بدون سرور اگرچه سروری برای مدیریت به کاربر اختصاص داده نمی‌شود اما این موضوع بدان معنا نیست که اصلاً سروری در کار نیست. در واقع مانند تمامی مدل‌های رایانشی در این جا هم سرور داریم، ولی تمامی تصمیمات مثل توزیع بار، تعداد اپ‌های روی سرور، انتخاب سرورها برای اجرای اپلیکیشن، کانفیگ CI/CD و ... بر عهده ارائه دهنده خدمات ابری است. اما برای درک نحوه کار یک پلتفرم بدون سرور، بهتر است با معماری آن آشنا باشیم. شکل ۲.۲ معماری یک پلتفرم خدمات ابری را نشان می‌دهد.

همانگونه که در تصویر مشخص است دوبخش کلی داریم، بخش لبه^۲ و بخش رییس^۳، بخش لبه شامل رابط گرافیکی کاربر^۴، Gateway API و Cloud Event Source می‌شود که برای تعامل با سرور رییس^۵ مناسب هستند. از طرف دیگر، در سرور رییس، درخواست‌ها ابتدا به صف رخدادها^۶ رسیده. صف رخدادها مسئول مدیریت رخداد

^۲edge

^۳master

^۴User Interface

^۵master server

^۶Event Queue

و نظم‌دهی به آن‌ها است. هر داده در صف رخداد نوبت‌دهی می‌شود و سپس به بخش توزیع‌کننده^۱ می‌رود. توزیع‌کننده آپلیکیشن را برای دیپلوی، یا درخواست را برای سرویس‌دهی به یک نود کارگر^۲ هدایت می‌کند نود کارگر نیز با ارسال کدهای پاسخ^۳ با سرور رییس در ارتباط است. [۵]

البته بهتر است بدانیم که امروزه ادبیات رییس-کارگر^۴ برای نامیدن این معماری منسوخ شده و به جای آن از ادبیات رییس-گره^۵ استفاده می‌کنند.

۳.۱.۲ ویژگی‌های پلتفرم‌های بدون سرور

امروزه پلتفرم‌های بدون سرور بسیاری وجود دارند که روزانه بر تعداد آن‌ها افزوده می‌شود. اما باید دنبال ویژگی‌هایی برای آن‌ها باشیم که براساس آن‌ها بتوان این پلتفرم‌ها را تفکیک کرد و دست به مقایسه‌ی آن‌ها زد. شاخص‌هایی که در این قسمت بررسی می‌کنیم شاخص‌های کمی و کیفی برای مقایسه‌ی بین این پلتفرم‌ها است.

۱. هزینه:

به طور معمول در رایانش بدون سرور، از مدل پرداخت پرداخت-به-ازای-استفاده^۶ استفاده می‌کنیم. یک ویژگی اساسی در تمایز بین ارائه‌دهندگان مختلف خدمات ابری، تفاوت آن‌ها در پشتیبانی از ویژگی scale-to-zero در این مدل محاسباتی^۷ است. این مورد باعث تفاوت معنی‌داری از هزینه‌ها در استفاده از پلتفرم‌های مختلف می‌شود. برخی از پلتفرم‌ها هم متن‌باز^۸ هستند که در این صورت، می‌توان به آسانی آن‌ها را بر روی ماشین مجازی یا سرور شخصی خود پیاده‌سازی کرد و برای استفاده از خدمات آن متحمل هیچ‌گونه هزینه‌ای نشد.

۲. کارایی و محدودیت‌ها^۹

^۱ dispatcher

^۲ worker node

^۳ Response Code

^۴ Master-worker

^۵ Master-Node

^۶ pay-as-you-go

^۷ Computational Model

^۸ Open Source

^۹ Performance and Limits

ارائه دهندگان مختلف، محدودیت‌های مختلفی را هم بر روی پلتفرم‌های مختلف خودشان اعمال می‌کنند. این محدودیت‌ها می‌توانند تعداد همزمان درخواست‌ها (number of concurrent requests)، استفاده از RAM و CPU توسط یک تابع، حداکثر زمان زنده ماندن بعد از اجرا توسط تابع و ... است. البته برخی از محدودیت‌ها را می‌توان با صرف هزینه یا خرید پلن‌های درآمدی سطح بالاتر برطرف کرد. مثلاً پلتفرم AWS Lambda functions می‌تواند با صرف هزینه‌ی بیشتری حداکثر تعداد درخواست را افزایش داد. درحالی‌که این مورد در پلتفرم‌های اوپن سورس وجود ندارد. در حالت کلی پلتفرم‌های متن بازی مثل openfaas محدودیت‌های بیشتری را برای کاربر اعمال می‌کنند. علت این امر می‌تواند این باشد که این پلتفرم‌ها از نظر تکنولوژی و بازدهی (performance) کاملاً از همتایان تجاری خود عقب هستند.

۳. زبان برنامه‌نویسی^۱

پلتفرم‌های بدون سرور از گستره‌ی عظیمی از زبان‌های برنامه‌نویسی پشتیبانی می‌کنند که شامل جاوااسکریپت، گو، پایتون، جاوا، سی‌شارپ، سوئیفت و پی‌اچ‌پی می‌شود. اکثر پلتفرم‌ها حداقل از ۵ زبان برنامه‌نویسی پشتیبانی می‌کنند. همچنین بسیاری از پلتفرم‌ها مستقل از زبان^۲ هستند. یعنی درحالی‌که در داخل کانتینر اجرا می‌شوند (مثلاً کانتینرهای داکر)، دیگر زبان برنامه‌نویسی برای آن‌ها اهمیتی ندارد. این پلتفرم‌ها توابع را در داخل کانتینر اجرا می‌کنند و نتیجه را برمی‌گردانند.

۴. مدل برنامه‌نویسی^۳

مدل‌های مختلفی برای تولید یک متد در پلتفرم‌های بدون سرور داریم. شیوه متداول استفاده از یک متد به نام main است که درون آن تابع اصلی تعریف می‌شود. همچنین معمولاً ورودی‌های تابع در قالب شیئی‌های json تعریف می‌شوند.

۵. ترکیب توابع^۴

روش‌های گوناگونی برای اینکه یک تابع یا جریان کاری پیچیده را پیاده سازی کنیم وجود دارد. یک روش

^۱Programming Languages

^۲Language Independent

^۳Programming Model

^۴Compositions

استفاده از ترکیب‌های توابع است. تا به حال ۷ ترکیب مختلف شناسایی شده است. پلتفرم‌های تجاری^۱‌هایی برای پیاده‌سازی این ترکیبات در خود تعبیه کرده‌اند. متاسفانه پلتفرم‌های متن باز مثل ۲ از این ترکیب پشتیبانی نمی‌کنند. در ادامه و در بخش کارهای مرتبط این ویژگی‌ها مطرح خواهند شد. کاربرد اصلی ترکیب توابع پیاده‌سازی عملکردهای پیچیده در پلتفرم بدون سرور است.

۶. استقرار^۳

پلتفرم‌ها سعی می‌کنند پیاده‌سازی‌ها در رایانش بدون سرور را تا حد ممکن ساده کنند. این یکی از دلایل به وجود آمدن این مدل رایانشی بوده. به صورت معمول پلتفرم‌ها برنامه‌ها را در قالب کانتینرهای داکری دریافت می‌کنند و درون کانتینر مربوطه کد را اجرا می‌کنند. علاوه بر داکر پلن‌هایی از جمله دریافت کد باینری، دریافت سورس کد و سپس کانینرایز کردن آن وجود دارد.

۷. امنیت و حسابداری^۴

این دو مورد در کنارهمدیگر به کار برده می‌شوند که معمولاً خارج از بحث‌های رایانش ابری کاملاً جدا از همدیگر به کار برده می‌شوند. در رایانش بدون سرور لازم است که اپ‌ها کاملاً از همدیگر جدا اجرا شوند به این دلیل که بتوانیم برای هر کاربر هزینه‌ای که باید پرداخت کند را محاسبه کنیم. در صورتی که اجرای کاربران از همدیگر تفکیک شده نباشد، محاسبه‌ی هزینه ممکن نیست. اما اجرای جداگانه‌ی توابع از یکدیگر علت دیگری نیز دارد، امنیت. لازم است که توابع جداگانه اجرا شوند تا در توابع و کاربران نتوانند در کارهای همدیگر دخالتی داشته باشند. این مورد حتی می‌تواند باعث به وجود آمدن باگ‌های امنیتی و دسترسی کاربران به سیستم کاربران دیگر از طریق مدل رایانشی ما شود.

۸. پایش و اشکال زدایی^۵

هر پلتفرم رایانشی امکاناتی از جمله پایش اولیه برای درخواست‌ها را به کاربر می‌دهد. البته این بحث یکی از مسائل باز در این حوزه است و نیاز به بررسی بیشتری دارد. در حال حاضر دیباگینگ از طریق تجزیه

^۱orchestrator

^۲openfaas

^۳Deployments

^۴Security and Accounting

^۵Monitoring and Debugging

و تحلیل لاگ‌های سیستم ممکن است ولی ممکن است در آینده بهبودهایی در این حوزه حاصل شود. علت اینکه دیباگینگ بسیار چالش برانگیز است این است که در رایانش بدون سرور اپ‌های ما کانتینرهای می‌شوند و چون محیط کانتیر محیطی ایزوله است، امکان مطالعه و دیباگینگ ممکن نیست. بعلاوه توابع تنها در حالت استفاده در پلتفرم زنده هستند؛ پس مدت زمان اشکال‌زدایی ما نیز بسیار محدود می‌شود. باید به این نکته دقت داشت که به طور متوسط در رایانش بدون سرور، توابع ۹۹ درصد زمان را از تاریخ استقرار روی سرور، در خواب هستند. اما در مورد پایش نرم‌افزار پلتفرم بدون سرور در داشبورد مدیریتی خود امکاناتی جهت مشاهده منابع مصرف شده، منابع آزاد مدت زمان استفاده‌شده، تعداد درخواست‌ها و فراخوانی‌ها، تعداد شروع‌های سرد و ... دارد. به‌علاوه ابزارهای پایش مانند prometheus به خوبی با این پلتفرم‌ها امکان اتصال دارند و با پنل‌هایی مانند grafana می‌توان از مانیتورینگ مضاعف برای این سرورها بهره برد.

۴.۱.۲ پلتفرم‌های تجاری

پلتفرم‌های اندکی برای این قسمت وجود دارد. معروف‌ترین آن‌ها عبارتند از : AWS Lambda Functions ،

IBM OpenWhisk و Microsoft Azure Functions ،Google Cloud Functions

۱. AWS Lambda Functions

پلتفرم AWS [۶] پلتفرم ارائه شده در بحث رایانش بدون سرور بود که دارای خلاقیت‌های بسیاری بود. از مدل برنامه‌نویسی، مدل هزینه‌ای، محدودیت منابع، امنیت و مانیتورینگ مخصوص خود استفاده می‌کند. همچنین AWS از زبان‌های Java، Node.js، Python و سی‌شارپ پشتیبانی می‌کند. این پلتفرم ارتباط خوبی با سایر خدمات و سرویس‌های AWS دارد و در این اکوسیستم اصطلاحاً حل شده است.

۲. Functions Cloud Google

پلتفرم شرکت گوگل با نام Google Cloud Functions [۷] به تازگی از حالت آلفا خارج شده. این سرویس از زبان‌های بسیاری ساپورت نمی‌کند ولی به خوبی به درخواست‌های HTTP و HTTPS پاسخ می‌دهد. در حال حاضر اگرچه عملکرد محدودی برای این پلتفرم شاهد هستیم ولی با توجه به سابقه گوگل و معماری

متفاوت این پلتفرم، آینده خوبی برای آن می‌توان متصور بود. این پلتفرم هنوز به خوبی با سرویس‌های رایانش ابری گوگل ارتباط برقرار نکرده و جای کار بیشتری دارد.

۳. Functions Azure Microsoft

پلتفرم بعدی، پلتفرم Microsoft Azure Functions [۸] است. این پلتفرم وب‌هوک‌های HTTP را برای تعامل با کاربر پیاده‌سازی کرده است. از زبان‌های Bash، PHP، Python، Node.js، سی‌شارپ و اف‌شارپ یا هر زبان اجرایی (چون از کانتینرهای داکری استفاده می‌کند) پشتیبانی می‌کند. بخشی از کدها و پروژه‌های انجام شده با این پلتفرم توسط میکروسافت در گیت‌هاب این پروژه متن‌باز شده‌اند. همچنین برای راحتی دیباگینگ میکروسافت در CLI مربوطه امکان Caching یا استفاده از حافظه موقت را گنجانده است. این پلتفرم به مقبولیت قابل قبولی در بین جوامع توسعه‌دهندگان رسیده و روز به روز بر امکانات آن افزوده می‌گردد.

۴. OpenWhisk Apache

پلتفرم آخر، پلتفرم OpenWhisk [۹] است که در برابر پلتفرم‌های دیگر البته بسیار ساده‌تر به نظر می‌رسد. این پلتفرم اپن‌سورس توسط شرکت IBM تولید و پشتیبانی می‌شود. از قابلیت استفاده زنجیره‌ای توابع بهره‌می‌برد و در مبحث Orchestration توابع از پلتفرم‌های رقیب خود جلوتر است (منبع به مقاله ۱). همچنین OpenWhisk توانایی اجرای هر تابعی را دارد؛ زیرا از داکر به عنوان runtime نیز استفاده می‌کند. سورس این پروژه در آدرس گیت‌هاب OpenWhisk موجود است. در شکل زیر نیز می‌توان معماری آن را مشاهده کرد.

همانگونه که در شکل بالا مشخص است. این معماری خیلی به معماری مینیمال یک پلتفرم بدون سرور شبیه است. البته در مقایسه با شکل قبل امکانات بیشتری از جمله امنیت، مانیتورینگ و لاگ‌گیری را اضافه کرده است.

۱. Apache OpenWhisk

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطر آنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد

نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

۲. Openfaas

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

۳. OpenLambda

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود

طراحی اساساً مورد استفاده قرار گیرد.

۵.۱.۲ پلتفرم‌های آزاد و متن باز

علاوه بر موارد فوق پلتفرم‌های متن‌بازی برای رایانش بدون سرور ارائه شده که در ادامه شرح خواهیم داد.

۱. پلتفرم Open Whisk

اگرچه این پلتفرم در قسمت قبل معرفی شد، اما به صورت متن‌باز وجود دارد و تنها IBM با پیاده‌سازی و ادغام در پلتفرم Bluemix که پلتفرم ابری شرکت IBM است، کسب درآمد می‌کند. این پلتفرم را به سادگی می‌توان بر روی ماشین‌های مجازی یا دستگاه‌های شخصی پیاده‌سازی کرد.

۲. پلتفرم Openfaas

پلتفرم بعدی، پلتفرم openfaas است که توسط جوامع آزاد توسعه داده شده است. پشتیبانی از زبان‌های جاوا، سی‌شارپ، پایتون و ... از ویژگی‌های آن است. برای کانترساز نیز از داکر به عنوان cli و موتور کانترینر سازی استفاده می‌کند. همچنین registry پیش‌فرض در این ابزار داکر رجستری است. جامعه‌ی رو به رشدی دارد و برای بسیاری از پروژه‌های کوچک و شرکت‌های متوسط مناسب است.

۳. پلتفرم Open Lambda

این پلتفرم از پلتفرم‌های جدید و متن‌باز است که تلاش می‌کند بسیاری از چالش‌ها و مسائل باز این حوزه را به طور خلاقانه‌ای حل کند. از ویژگی‌های پلتفرم open lambda میتوان به زمان اجرای سریع تر تابع ها به خاطر زمان شروه بهتری نسبت به سایر نمونه ها نام برد. همچنین از توابع state-ful هم پشتیبانی می‌کند. همچنین استفاده از توابع بدون سرور با دیتابیس ها و دیباگینگ بهتر فراهم شده است. [۱۰]

۶.۱.۲ Function-as-a-Service

واژه‌ی بعدی که در این حوزه بسیار مطرح می‌شود واژه‌ی Function-as-a-Service یا به اختصار FaaS است.

FaaS یک دسته‌بندی جدید در سرویس‌های رایانشی است که با استفاده از یک پلتفرم بدون سرور، به اجرا، توسعه یا مدیریت توابع، بدون هیچ‌گونه پیچیدگی خاص یا نگرانی برای نگهداری زیرساخت که برای استقرار و پیاده‌سازی یک اپلیکیشن که سابقاً و در مدل‌های غیر بدون سرور، باید کانفیگ می‌کرده ایم.

تولید یک نرم افزار بر اساس FaaS، روشی برای رسیدن به یک مدل رایانشی بدون سرور است و در قالب توسعه‌ی میکروسرویس‌ها و نانوسرویس‌هایی در توابع، به‌دست می‌آید. از آنجایی که FaaS به صورت حین تقاضا^۱ به ما خدمات می‌دهد برای توسعه‌ی خدماتی که به تجزیه و تحلیل داده نیاز دارند مانند سرویس‌های اینترنت اشیا، برنامه‌های موبایل و وب اپلیکیشن‌ها بسیار کاربرد دارد.

حال می‌توان به مقایسه بین رایانش بدون سرور با FaaS پرداخت، بر اساس تعریف FaaS را می‌توان یک پیاده سازی از رایانش بدون سرور نامید. همچنین رایانش بدون سرور منتهی به اجرای توابع تحت سرویس نمی‌شود. بلکه حوزه های وسیع تری از جمله Mobile-Backend-as-a-Service یا درموردی PaaS را شامل می‌شود.

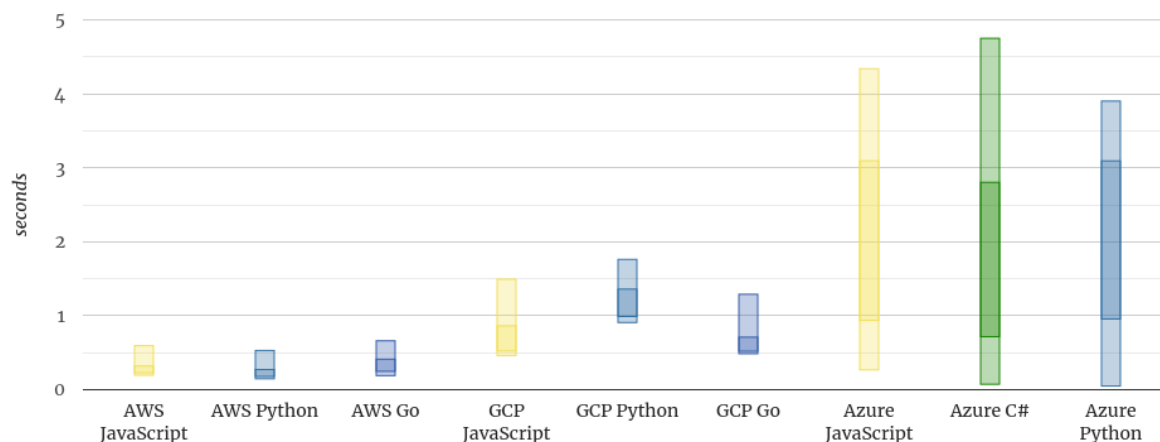
۲.۲ Scale-to-Zero

یکی از ویژگی‌های کلیدی در رایانش بدون سرور، قابلیت Scale-to-Zero است. این قابلیت موجب پیاده‌سازی پلن‌های هزینه مانند پلن هزینه ردخت حین مصرف^۲ می‌شود. قابلیت scale-to-zero یعنی اینکه در پلتفرم ما هرگاهی که تابع مدت زیادی بلا استفاده باشد، منابع پردازشی آن گرفته می‌شوند و آن تابع اصطلاحاً zero-scale می‌شود تا تابع دیگر فعال نباشد. اگر از دید ارائه دهنده خدمات به این موضوع نگاه کنیم، برای ما این فایده را دارد که منبع بلا استفاده ما در این حالت آزاد می‌شود و آن منبع (CPU و RAM) به کانتینر دیگری که می‌خواهد استفاده شود اختصاص می‌یابد. طبق آمار توابع در FaaS به ندرت صدا زده می‌شوند. ۸ طبقه‌بندی برای فراخوانی توابع انجام شده و مشخص شده که ۵۰٪ توابع زیر ۱ ثانیه و ۹۶٪ آن‌ها زیر ۹ ثانیه اجرا می‌شوند [۴]. بنابراین اگر منبع به یک تابع به مدت زیادی اختصاص یافته باشد دچار اتلاف منابع زیادی خواهیم شد.

از دید کاربر هم اگر بخواهیم به موضوع نگاه کنیم. Scale-to-Zero باعث فراهم شدن ویژگی پرداخت حین استفاده می‌شود. این صرفه اقتصادی بزرگی برای ما دارد. فرض کنید یک اپلیکیشن در حال اجرا داریم که اجرای

^۱ on-demand

^۲ pay-as-you-go



شکل ۳.۲: اهمیت زبان و پلتفرم بدون سرور در تأخیر شروع سرد

انفجاری دارد و این اپلیکیشن به ندرت اجرا می‌شود. اگر بخواهیم از مدل‌های قدیمی پرداخت استفاده کنیم، باید هزینه زیادی صرف کنیم، در حالی که در اکثر اوقات تابع ما هم بلا استفاده است. در حالی که با مدل پرداخت در سیستم‌های بدون سرور، این مورد بسیار برای ما به صرفه می‌شود.

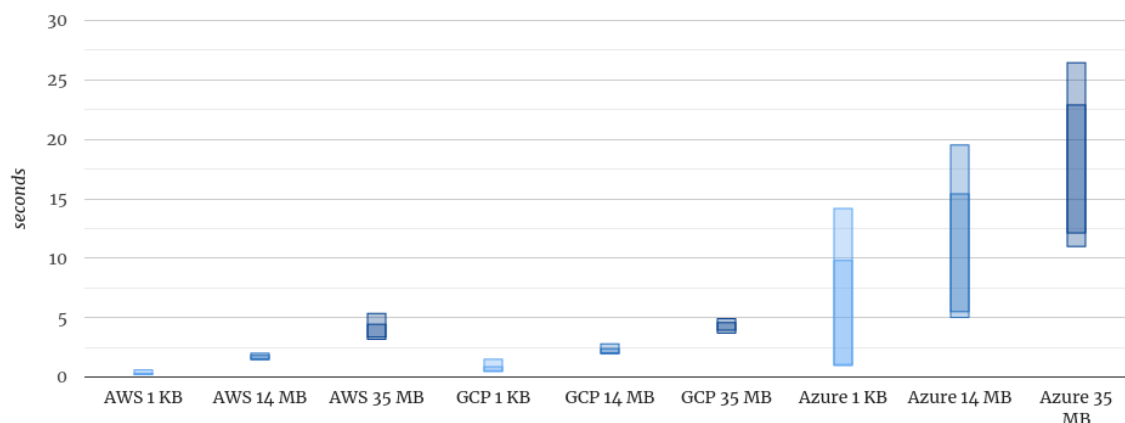
دو مورد بالا از مزایای قابلیت Scale-to-Zero در سیستم‌های بدون سرور هستند. اما در این صورت با یک چالش جدی به نام تأخیر شروع سرد هم مواجه خواهیم شد که در ادامه به آن می‌پردازیم.

۳.۲ تأخیر شروع سرد

در قسمت قبل به بیان ویژگی Scale-to-Zero در پلتفرم‌های بدون سرور و مزایای آن پرداختیم. اما این ویژگی معایبی هم دارد. یکی از مهم‌ترین معایب آن، تأخیر شروع سرد^۱ است.

بگذارید تأخیر شروع سرد را در قالب یک مثال بیان کنیم. فرض کنید پس از مدتی بلا استفاده بودن تابع منابع آن گرفته شده و اصطلاحاً سرد شده است. حال یک درخواست برای اجرا برای آن تابع سرد شده می‌رسد. در حالی که تابع ما منبعی ندارد و اجرا نمی‌شود؛ آن درخواست باید مدتی منتظر بماند تا تابع مورد نظر ما دوباره آماده شود و در سرور بارگذاری شود. این آماده سازی، مراحل مختلفی دارد که در تصویر ۳.۲ نشان داده شده است.

^۱Cold Start Latency



شکل ۴.۲: اهمیت اندازه کتابخانه برنامه در تأخیر شروع سرد

همانگونه که در تصویر ۳.۲ می‌بینیم، این آماده سازی شامل آماده سازی کانتینرها، آماده سازی تابع، اختصاص منابع به کانتینر و در نهایت، اجرا در پلتفرم است. این تأخیر بسیار قابل توجه است و در شکل ۱.۱ نیز به خوبی نشان داده شده است.

همچنین، یکی از مواردی که شروع سرد اتفاق می‌افتد، فراخوانی تابع برای اولین بار در پلتفرم است. حال چه عواملی در شروع سرد نقش دارند؟ عوامل عمده‌ای در این کار دخیل هستند ولی مهم‌ترین آن‌ها زبان برنامه‌نویسی و نوع پلتفرمی که در آن کد را اجرا می‌کنیم و کانفیگ نرم‌افزاری و سخت‌افزاری آن پلتفرم است. در مورد اهمیت زبان‌های برنامه‌نویسی می‌توان گفت چون زبان‌های مختلف زمان اجراهای متفاوتی دارند بنابراین موثر هستند. شکل زیر اهمیت زبان‌های برنامه‌نویسی را نشان می‌دهد. در این شکل از پلتفرم‌های مختلف برای اجرای توابع مختلف برای محاسبه‌ی زمان اجرا با در نظر گرفتن تأخیر شروع سرد استفاده شده است. در آزمایش دیگری با جاوا اسکریپت تابع‌های یکسانی نوشته شده با این تفاوت که حجم کتابخانه‌های هر تابع متفاوت است. این آزمایش در پلتفرم‌های مختلف نیز انجام شده و نتیجه طبق شکل زیر رسم شده است. بنابراین با توجه به شکل ۴.۲ می‌توان نتیجه گرفت حجم کتابخانه توابع نیز در مدت زمان تأخیر شروع سرد بسیار موثر است.

فصل ۳

کارهای مرتبط

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

فصل ۴

نتیجه‌گیری

در فصل قبل در ارتباط با راهکارهای ارائه شده توسط هر یک از مقاله‌ها به تفصیل، تشریح نمودیم. در این فصل می‌خواهیم به بیان نتایج و مقایسه راهکارهای ارائه شده بپردازیم. در ابتدا به مقایسه‌های روش‌های مبتنی بر مکانیزم DRX/DTX با یکدیگر می‌پردازیم.

مراجع

- [1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol.62, p.44-54, Nov. 2019.
- [2] P.-M. Lin and A. Glikson, "Mitigating cold starts in serverless platforms: A pool-based approach," *arXiv preprint arXiv:1903.12221*, 2019.
- [3] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, "Agile cold starts for scalable serverless," in *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.
- [4] M. Shahradd, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 {USENIX} Annual Technical Conference ({USENIX} {ATC} 20)*, pp.205-218, 2020.
- [5] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*, pp.1-20, Springer, 2017.
- [6] A. AWS, "Aws lambda functions," <https://aws.amazon.com/lambda/>.
- [7] G. INC., "Google cloud functions," <https://cloud.google.com/functions>.
- [8] Microsoft, "Microsoft azure platform," <https://azure.microsoft.com/en-us/services/functions/>.
- [9] A. openwhisk corporation, "Ibm openwhisk platform," <https://openwhisk.apache.org/>.
- [10] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *8th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.