APJ 6 (4):57-65, Winter 2025

Arman Process Journal (APJ)

Homepage: https://www.armanprocessjournal.ir

Qualitative Approach to Improve Distribution Systems with Microservice Architecture

Z. Sobhanipoor [1], A. Tahriri [2]

[1,2] Department of Computer engineering , Qo.C., Islamic Azad University , Qom, Iran

RESEARCH PAPER

Received: 2025-9-22
Accepted: 2025-11-7

KEYWORDS:
Microservice Architecture,
System Quality Optimization,
Distribution Systems,
Integrated Migration,
Service Based Analysis,
Deployment,

[1] Corresponding author:
z.sobhanipoor@iau.ac.ir

ABSTRACT

Rapid development of cloud computing technologies and a significant increase in demand for scalable software systems has forced organizations to review their distributed systems architecture .traditional monolithic architecture , despite its initial efficiency , is faced with serious challenges in the field of maintenance , scalability , and independent deployment because of the strong connection between components .this paper reviews new approaches to improve distribution systems with focus on microservice architecture .in this research , a comprehensive four phase framework (discovery and analysis , service identification , design and implementation , and implementation) is presented that guides the migration process from integrated architecture to microservice systematically .the proposed framework combines static and dynamic analyses of support system , using graph - based learning techniques to identify the boundaries of service , and use architecture patterns , enables organizations to experience successful migration by maintaining service quality and reducing operational risks .evaluating the framework on a real financial system (UVT) shows that the application of the proposed approach reduces the response time by 42 % , reduces the error rate to less from 0.5 % and improves the error recovery time by 65 % .this research presents practical guidelines and evaluation of existing methodologies , a way for researchers and software engineers in the field of migration to microservice architecture.

نشریه تخصصی آرمان پردازش، دوره ۶، شماره ۴، سال ۱۴۰۴

فصلنامه تخصصی آرمان پردازش
(APJ)
Homepage: www.armanprocessjournal.ir

آرمان پردازش APJ
فصلنامه تخصصی فناوری اطلاعات و ارتباطات
شماره مجوز: ۸۷۰۹۰

رویکرد کیفیت‌محور جهت بهسازی سیستم‌های توزیعی با معماری میکروسرویس

زهرا سبحانی پور[۱]، علیرضا تحریری[۲]

[۱و۲] گروه مهندسی کامپیوتر، واحد قم، دانشگاه آزاد اسلامی، قم، ایران

چکیده

مقاله پژوهشی

پیشرفت سریع فناوری‌های محاسباتی ابری و افزایش چشمگیر تقاضا برای سیستم‌های نرم‌افزاری مقیاس‌پذیر، سازمان‌ها را ناگزیر به بازنگری در معماری سیستم‌های توزیعی خود ساخته است. معماری یکپارچه سنتی، علی‌رغم کارایی اولیه، به دلیل اتصال محکم میان مؤلفه‌ها، با چالش‌های جدی در زمینه قابلیت نگهداری، مقیاس‌پذیری و استقرار مستقل مواجه است. این مقاله به بررسی رویکردهای نوین در بهسازی سیستم‌های توزیعی با تمرکز بر معماری میکروسرویس می‌پردازد. در این تحقیق، یک چارچوب جامع چهارمرحله‌ای (کشف و تحلیل، شناسایی سرویس‌ها، طراحی و پیاده‌سازی، و استقرار و نظارت) ارائه شده است که فرآیند مهاجرت از معماری یکپارچه به میکروسرویس را به‌صورت نظام‌مند هدایت می‌کند. چارچوب پیشنهادی با ترکیب تحلیل‌های ایستا و پویا از سیستم پشتیبان، استفاده از تکنیک‌های یادگیری ماشین مبتنی بر گراف برای شناسایی مرزهای سرویس، و بهره‌گیری از الگوهای معماری رویدادمحور، به سازمان‌ها امکان می‌دهد تا با حفظ کیفیت سرویس و کاهش ریسک‌های عملیاتی، مهاجرتی موفق را تجربه کنند. ارزیابی چارچوب بر روی یک سیستم واقعی مالی (UVT) نشان می‌دهد که به‌کارگیری رویکرد پیشنهادی، زمان پاسخ را تا ۴۲٪ کاهش، نرخ خطا را به کمتر از ۰٫۵٪ محدود و زمان بازیابی خطا را تا ۶۵٪ بهبود می‌بخشد. این پژوهش با ارائه دستورالعمل‌های عملی و ارزیابی متدولوژی‌های موجود، نقشه راهی برای پژوهشگران و مهندسان نرم‌افزار در زمینه مهاجرت به معماری میکروسرویس فراهم می‌آورد.

واژگان کلیدی:

معماری میکروسرویس، بهسازی کیفیت سیستم، سیستم‌های توزیعی، مهاجرت یکپارچه، تحلیل سرویس‌محور، استقرار،

مقدمه

در دهه اخیر، پیچیدگی سیستم‌های نرم‌افزاری، اندازه و هزینه‌های مرتبط با آن‌ها به‌طور چشمگیری افزایش یافته است [۱]. هم‌چنین، رقابت شدید میان سازمان‌ها، آن‌ها را مجبور به انجام به‌روزرسانی‌های سریع سیستم‌ها می‌کند و در عین حال باید اطمینان حاصل کنند که سیستم‌هایشان کاملاً عملیاتی باقی می‌مانند. معماری میکروسرویس[1] به‌عنوان یک پارادایم محاسباتی برجسته در مهندسی نرم‌افزار ظهور کرده است و شرکت‌ها به‌طور فزاینده‌ای سیستم‌های یکپارچه خود را برای بهبود عملکرد دیجیتال و رقابت‌پذیری بازسازی می‌کنند. معماری یکپارچه اگرچه در اجرای منطق‌های پیچیده و مهم تجاری در یک دوره زمانی طولانی کارایی نشان داده است [۲]، اما به دلیل اتصال محکم بین مؤلفه‌های داخلی، تغییر در عملکرد یک ماژول اغلب نیازمند تغییرات در ماژول‌های متعدد دیگر است که تلاش و توجه لازم برای توسعه را افزایش می‌دهد [در مقابل، معماری مبتنی بر میکروسرویس امکان توسعه برنامه‌های کاربردی پیچیده و میان‌سازمانی را از طریق یکپارچه‌سازی سرویس‌های قابل استفاده مجدد، نسبتاً خودمختار، عموماً ناهمگن و توزیع‌شده فراهم می‌کند [۳].

میکروسرویس ها یک پیشرفت جدید هستند که چند سال پیش در معرض دید قرار گرفتند. آنها در مقایسه با معماری های یکپارچه قدیمی مزایای بسیاری را ارائه می دهند. به همین دلیل است که بسیاری از شرکت های بزرگ فناوری با موفقیت به میکروسرویس ها روی آورده اند. در حال حاضر، معماری یکپارچه مدل پیش فرض برای ایجاد یک برنامه نرم افزاری است. روند آن در حال کاهش است زیرا نمی تواند با خواسته ها و چالش های برنامه های کاربردی جدید که اکنون بسیار بزرگ و پیچیده هستند، سازگاری داشته باشند. در معماری یکپارچه، کاربرد به عنوان یک واحد تقسیم ناپذیر ساخته می شود. این معمولاً به این معنی است که برنامه دارای سه جزء اصلی است که اطلاعات را با یکدیگر مبادله می کنند: یک رابط کاربر، یک سمت سرور و یک پایگاه داده. این معماری با یک پایه کد بزرگ مشخص می شود و تقریباً هیچ ماژولاریتی ندارد. از آنجا که آنها یک پایه کد واحد دارند، می توانند بسیار بزرگ شوند و از این رو نگهداری آنها دشوار است. کل برنامه باید با یک تغییر کوچک در کد دوباره مستقر شود. مهم‌تر این واقعیت است که خیلی قابل اعتماد نیست زیرا یک اشکال در هر بخشی از کد می‌تواند کل برنامه را از بین ببرد [۴].

با این حال، معماری یکپارچه دارای مزایای ظریفی است و با برخی تغییرات هنوز هم می تواند برای بسیاری از برنامه های مدرن مفید باشد. این موارد عبارتند از: سهولت استقرار (از آنجایی که فقط یک فایل نیاز به استقرار دارد)، سهولت توسعه (در مقایسه با میکروسرویس ها) و تأخیر شبکه و امنیت که در معماری میکروسرویس ها بیشتر قابل توجه است. آزمایش معماری یکپارچه نیز بسیار آسان است. ما می توانیم این کار را به سادگی با راه اندازی برنامه و آزمایش رابط کاربری با سلنیوم انجام دهیم. با این حال، برخی از ایرادات این معماری، تغییر به میکروسرویس ها را به یک ضرورت تبدیل کرده است. از آنجایی که برنامه های امروزی بزرگ و پیچیده هستند، برای مفید بودن، باید قوی و قابل اعتماد باشند. منابع باید به طور مؤثر مورد استفاده قرار گیرند تا کاربران بتوانند در حین گشت و گذار در برنامه، تجربه یکپارچه داشته باشند. بسیاری از اجزای برنامه ممکن است نیازهای منابع متفاوتی داشته باشند. برخی ممکن است به چرخه های CPU بیشتری نیاز داشته باشند، برخی دیگر به حافظه بیشتر و غیره نیاز دارند. مقیاس بندی در معماری یکپارچه با ایجاد کپی از برنامه انجام می شود. این به این معنی است که همه این کپی ها به تمام داده ها دسترسی خواهند داشت که به نوبه خود باعث کاهش اثربخشی حافظه پنهان و افزایش مصرف حافظه و ترافیک ورودی/خروجی می شود [۵].

خدمات میکرو، مجموعه ای از سرویس های کوچک است که حول قابلیت های تجاری می چرخد و به عنوان مستقل، غیرمتمرکز و قادر به نوشتن در زبان های برنامه نویسی مختلف مشخص می شود. معماری سنتی یکپارچه به صورت یک واحد ساخته شد که مقیاس آن دشوار بود و برای توسعه فرآیندهای پیچیده مناسب نبود. نیاز به یک الگوی قوی، انعطاف پذیر و قابل اعتماد، منجر به رشد و محبوبیت خدمات خرد شد. از آنجایی که خدمات خرد به عنوان گروهی از خدمات ساخته می شوند، به یک مکانیسم ارتباطی صریح نیاز دارند. این ارتباط با کمک رابط های برنامه نویسی کاربردی (API) انجام می شود که ستون فقرات این الگوی معماری را تشکیل می دهند. مبنای اصلی محبوبیت خدمات خرد مزایای متعددی است که ارائه می دهد که شامل ساختار مدولار، استقرار مستقل، سهولت یکپارچه سازی و اصلاح، جداسازی خطا و تحویل مداوم است. اما به دلیل ماهیت توزیع شده آن، تمرکز بر مکانیسم های ارتباطی کارآمد و طراحی خدمات برای به حداقل رساندن پیچیدگی های عملیاتی مهم است [۶].

با این حال، مهاجرت برنامه‌های کاربردی یکپارچه به میکروسرویس فرآیندی پیچیده است که نیازمند بررسی جنبه‌های متعددی از جمله رویکرد مهاجرت، روش شناسایی میکروسرویس‌ها و کیفیت سرویس‌های کشف‌شده است [۸-۷]. به‌ویژه، شناسایی میکروسرویس‌ها به‌عنوان

[1] Microservice Architecture (MSA)

مهم‌ترین و زمان‌برترین مرحله در فرآیند مهاجرت شناخته می‌شود . بهسازی سیستم‌های توزیعی در این زمینه، به مجموعه فعالیت‌هایی اطلاق می‌شود که با هدف ارتقای کیفیت معماری، کاهش وابستگی‌های ناخواسته، بهبود عملکرد و افزایش قابلیت اطمینان سیستم‌های توزیع‌شده انجام می‌گیرد.

در حالی که پژوهش‌های متعددی در زمینه مهاجرت به میکروسرویس انجام شده است، شکاف‌های پژوهشی قابل توجهی وجود دارد. بسیاری از رویکردهای موجود بر جنبه‌های فنی مهاجرت متمرکز شده‌اند و کمتر به استراتژی‌های جامع کیفیت‌محور که ابعاد سازمانی، عملیاتی و معماری را یکپارچه می‌سازد، پرداخته‌اند [۹][۱۰]. علاوه بر این، ارزیابی چارچوب‌های تجزیه اغلب به دلیل سیستم‌های معیار ناسازگار، معیارهای نامتناسب و تکرارپذیری محدود، پراکنده است [۱۱]. هدف این مقاله ارائه یک رویکرد جامع و کیفیت‌محور برای بهسازی سیستم‌های توزیعی با معماری میکروسرویس است که هم به چالش‌های فنی و هم به الزامات سازمانی مهاجرت می‌پردازد.

ساختار مقاله حاضر به این شرح است: بخش ۲ به مرور تحقیقات مرتبط می‌پردازد. بخش ۳ چارچوب پیشنهادی را تشریح می‌کند. بخش ۴ شامل بحث و مقایسه رویکردهاست. بخش ۵ نتیجه‌گیری و بخش ۶ مسیرهای آتی تحقیق را ارائه می‌دهد.

تحقیقات مرتبط

میکروسرویس‌ها نوعی معماری هستند که از معماری‌های سرویس‌گرا تکامل یافته‌اند. ایده اصلی، سازماندهی سیستم‌ها از طریق استفاده از مؤلفه‌های خودمختار کوچک است که صرفاً از طریق تبادل پیام ارتباط برقرار می‌کنند. بر خلاف سیستم‌های یکپارچه و معماری‌های سنتی سرویس‌گرا، MSA بر استقلال، مقیاس‌پذیری و انسجام معنایی هر مؤلفه تأکید بیشتری دارد [۱۲]. در نتیجه، میکروسرویس‌ها را می‌توان به زبان‌های برنامه‌نویسی مختلف توسعه داد، از پایگاه‌های داده گوناگون استفاده کرد و مستقل از ساختار بقیه سیستم آزمایش نمود [۱۳]. میکروسرویس‌ها از پروتکل‌های ارتباطی سبک مانند HTTP استفاده می‌کنند و همچنین می‌توانند به‌طور غیرمستقیم از طریق کارگزارهای پیام مانند Kafka و RabbitMQ ارتباط برقرار نمایند. استقرار میکروسرویس‌ها می‌تواند بر روی پلتفرم‌های اجرایی مختلف یا کانتینرهای سبک مانند Docker انجام شود [۱۴].

از نظر ویژگی‌های کیفیت نرم‌افزار، پیاده‌سازی میکروسرویس‌ها (MSA) باعث ارتقای قابلیت همکاری و استفاده مجدد از سیستم‌های نرم‌افزاری پیچیده، تسهیل مقیاس‌پذیری و بهبود قابلیت نگهداری می‌شود. شرکت‌های موفقی مانند نتفلیکس، آمازون و اسپاتیفای توانسته‌اند معماری‌های میکروسرویسی موفقی ایجاد کنند. با این حال، اگرچه مزایای استفاده از معماری میکروسرویس در طراحی سیستم‌های پیچیده بسیار است، مهاجرت از معماری یکپارچه به معماری میکروسرویس همواره ساده نیست . یکی از اشتباهات رایج، ایجاد به اصطلاح "یکپارچه توزیع‌شده " است؛ سیستمی که در آن سرویس‌ها به قدری به هم متصل هستند که یک مشکل در یک سرویس می‌تواند کل سیستم را مختل کند [۱۵]. این وضعیت زمانی رخ می‌دهد که به جای تمرکز بر هماننداسازی مناسب و ارتباطات ناهمگام، سرویس‌ها به صورت همگام و نقطه‌به‌نقطه به یکدیگر متصل شوند. جدول ۱ خلاصه‌ای از مزایا و چالش‌های کلیدی معماری میکروسرویس را ارائه می‌دهد.

جدول ۱: مزایا و چالش‌های کلیدی معماری میکروسرویس

مؤلفه	مزایا	چالش‌ها
مقیاس‌پذیری	استقرار مقیاس‌پذیر مستقل هر سرویس	مدیریت بار در شبکه پیچیده
استقرار	استقرار مستقل و فراوان، جایگزینی آسان	هماهنگی استقرار بین سرویس‌ها
فناوری	استفاده از زبان‌ها و پایگاه‌های داده متنوع	سازگاری نسخه‌هایAPI
سازمانی	تیم‌های کوچک و مستقل(DevOps)	هزینه ارتباطات بین تیمی
قابلیت اطمینان	جداسازی خطا، تأثیر محدود خطا	تراکنش‌های توزیع‌شده (هماهنگی)
مدیریت	راحت‌تر در ابعاد کوچک	پیچیدگی در نظارت و دیباگ توزیع‌شده

استراتژی‌های مهاجرت و شناسایی میکروسرویس

مهاجرت از یک معماری یکپارچه به میکروسرویس به طور کلی از سه استراتژی اصلی پیروی می‌کند: رویکرد بیگ بنگ که در آن کل سیستم از نو نوشته می‌شود (پُرخطر)، رویکرد لایه‌لایه که به تدریج و با جایگزینی قطعات قدیمی توسط سرویس‌های جدید انجام می‌گیرد (کم‌خطرتر و رایج‌تر) و رویکرد جداسازی مبتنی بر مؤلفه که بر اساس وابستگی‌های کد انجام می‌شود [۱۶]. شناسایی میکروسرویس‌ها (یعنی تصمیم‌گیری درباره مرزها و دانه‌بندی سرویس‌ها) به عنوان بحرانی‌ترین مرحله در فرآیند مهاجرت شناخته می‌شود. پژوهش‌های متعددی رویکردهای مختلفی را برای این منظور پیشنهاد کرده‌اند که می‌توان آن‌ها را در چهار دسته اصلی طبقه‌بندی کرد:

- رویکردهای مبتنی بر تحلیل ایستا : این رویکردها با تحلیل وابستگی‌های کد (مانند گراف تماس‌های تابعی یا وابستگی بسته‌ها، کتابخانه‌ها) سعی در یافتن خوشه‌های با انسجام بالا و وابستگی کم (همبستگی پایین) دارند. روش‌های

خوشه‌بندی سلسله‌مراتبی مانند HDBScan در این زمینه نتایج متوازنی نشان داده‌اند .

- رویکردهای مبتنی بر تحلیل پویا : این رویکردها جریان اجرای سیستم در زمان واقعی را بر اساس تراکنش‌ها و داده‌های واقعی (مانند لاگ‌ها) بررسی می‌کنند تا تعاملات معنایی بین مؤلفه‌ها را کشف نمایند.
- رویکردهای ترکیبی : ترکیبی از تحلیل ایستا و پویا برای دستیابی به نتایج دقیق‌تر.
- رویکردهای مبتنی بر یادگیری ماشین : روش‌های پیشرفته‌تر که از تکنیک‌هایی مانند مدل‌سازی موضوع ، پردازش زبان طبیعی (NLP) برای شناسایی مفاهیم تجاری و اخیراً از شبکه‌های عصبی گرافی (GNNs) برای مدل‌سازی وابستگی‌های ساختاری سیستم مانند روش MAGNET استفاده می‌کنند.

یک مطالعه نقشه‌برداری سیستماتیک اخیر نشان می‌دهد که علی‌رغم تنوع روش‌ها، "شناسایی میکروسرویس" هنوز به عنوان یک مانع بزرگ در تلاش‌های مهاجرت سیستم‌ها باقی مانده است و نیاز به تحقیقات بیشتر برای توسعه تکنیک‌های شناسایی مؤثر، به ویژه آنهایی که نقش‌ها و وابستگی‌ها را در یک معماری میکروسرویس در نظر می‌گیرند، احساس می‌شود.

کیفیت و معماری رویدادمحور در میکروسرویس‌ها

کیفیت نرم‌افزار در معماری میکروسرویس به شش ویژگی کلیدی وابسته است :مقیاس‌پذیری، عملکرد، در دسترس بودن، مدیریت‌پذیری، امنیت و آزمون‌پذیری [۷]. پژوهش‌های اخیر بر رویکردهای "کیفیت‌محور" (Quality-Driven)در مهاجرت تأکید دارند، جایی که محدودیت‌های کیفیت (مانند زمان پاسخ، توان عملیاتی) موتور محرک اصلی مهاجرت هستند. آنتی‌الگوهای رایج در میکروسرویس‌ها (مانند سرویس خدای داده، زنجیره سخت و ...) به عنوان شاخص‌هایی برای جهت‌دهی فرآیند مهاجرت استفاده می‌شوند [۳][۱۰].

از سوی دیگر، یکی از بزرگ‌ترین چالش‌ها در سیستم‌های توزیع‌شده، مدیریت تراکنش‌های توزیع‌شده و تضمین سازگاری داده‌ها در سرویس‌هایی است که پایگاه داده جداگانه دارند [۵]. معماری رویدادمحور[1] به عنوان راه حل کلیدی برای این چالش و کاهش وابستگی‌های مضر مطرح است [۷]. به جای فراخوانی مستقیم (نقطه-به-نقطه سخت)، سرویس‌ها "رویداد"هایی را در مورد کارهایی که انجام داده‌اند منتشر می‌کنند. الگوی Saga اساساً چارچوبی برای پیاده‌سازی تراکنش‌های بلندمدت در این محیط توزیع‌شده با استفاده از دو روش اصلی است :هماهنگی که در آن یک هماهنگ کننده مرکزی جریان را کنترل می‌کند (ساده‌تر برای دیباگ، اما خطر ایجاد نقطه شکست واحد) و نقشه‌مندی که در آن سرویس‌ها به رویدادها واکنش نشان می‌دهند (کاهنده وابستگی، اما پیچیده‌تر برای ردیابی).

ادراک از میکروسرویس به عنوان یک راه حل جادویی برای همه مشکلات، منجر به نادیده گرفتن پیچیدگی‌های ذاتی آن شده است. بدون طراحی دقیق مرزها و استفاده از الگوهای ناهمگام، مزیت "استقلال" سرویس از بین رفته و سیستم به "یکپارچه توزیع‌شده" بدل می‌شود که هم پیچیدگی سیستم‌های توزیع‌شده و هم شکنندگی سیستم‌های یکپارچه را دارد.

رویکرد پیشنهادی: چارچوب جامع بهسازی

در این بخش، چارچوب پیشنهادی خود برای بهسازی سیستم‌های توزیعی به سمت معماری میکروسرویس ارائه می‌شود. این چارچوب که با الهام از فرآیند SPReaD [۶] و رویکردهای کیفیت‌محور [۳][۱۰] طراحی شده، شامل چهار فاز اصلی است :کشف و تحلیل، شناسایی سرویس‌ها، طراحی و پیاده‌سازی و استقرار و نظارت.

فاز اول: کشف و تحلیل سیستم پشتیبان (Reverse Engineering)

هدف در این فاز، درک عمیق سیستم یکپارچه موجود است.

- مهندسی معکوس کد :(Code Reverse Engineering) استخراج گراف وابستگی از کد منبع. این کار شامل شناسایی کلاس‌ها، متدها، تماس‌ها، و وابستگی‌های پایگاه داده می‌شود.
- تحلیل پویا :(Dynamic Analysis) با اجرای سناریوهای تجاری معمول (مانند تراکنش خرید، ورود کاربر)، لاگ‌های اجرا ثبت و مسیرهای اجرایی استخراج می‌شوند.
- مدل‌سازی مفهومی :(Domain Modeling) با کمک تیم کسب و کار، مدل حوزه به دست آمده و کاندیدهای اولیه برای مرزهای سرویس مشخص می‌شوند.

خروجی این فاز یک مدل گرافی یکپارچه از سیستم است که در آن گره‌ها مؤلفه‌ها و یال‌ها نشان‌دهنده انواع وابستگی‌ها (ساختاری، اجرایی، معنایی) هستند.

فاز دوم: شناسایی هوشمندانه سرویس‌ها

این فاز قلب چارچوب پیشنهادی است و از ترکیبی از رویکردهای مبتنی بر متادیتا و یادگیری ماشین استفاده می‌کند [۱][۴][۸]. به جای استفاده از یک الگوریتم صرف، ما یک خط لوله تصمیم‌گیری چند مرحله‌ای پیشنهاد می‌کنیم:

- خوشه‌بندی ساختاری اولیه :با استفاده از الگوریتم HDBScan بر روی گراف وابستگی ایستا خوشه‌های اولیه

[1] Event-Driven Architecture (EDA)

مبتنی بر مؤلفه‌های سخت‌افزاری/کدی شناسایی می‌شوند . این مرحله "انسجام فنی" را تضمین می‌کند.

- تحلیل معنایی :(Semantic Analysis) با استفاده از تکنیک‌های پردازش زبان طبیعی (NLP) روی نام کلاس‌ها، متدها و کامنت‌ها، مفاهیم تجاری استخراج می‌شوند [۸]. این کار به شناسایی سرویس‌های مبتنی بر "قابلیت تجاری" کمک می‌کند.
- بهبود با شبکه عصبی گرافی :(GNN) در این مرحله از مدل MAGNET] [۸] برای اصلاح مرزها استفاده می‌شود GNN .با در نظر گرفتن ویژگی‌های ساختاری (وابستگی‌ها) و معنایی همزمان، پیشنهاد می‌دهد که آیا دو مؤلفه مجاور باید در یک سرویس قرار گیرند یا از هم جدا شوند.
- ارزیابی و اعتبارسنجی کیفی :سرویس‌های شناسایی‌شده بر اساس معیارهای ارزیابی کلیدی (جدول ۲) رتبه‌بندی می‌شوند.

جدول ۲: معیارهای ارزیابی کیفیت تجزیه

معیار	فرمول (خلاصه)	توضیح	مطلوبیت
چگالی درونی(Cohesion)	نسبت اتصالات داخلی به کل اتصالات	میزان خودمختاری و تمرکز سرویس	بالا
وابستگی بین سرویس(Coupling)	تعداد تماس‌های همگام خروجی	استقلال و کاهش ریسک اثرات جانبی	پایین
قابل فهم بودن رابط(IFN)	تعداد و اندازه متدهایAPI	سهولت استفاده و توسعه	پایین
توزیع اندازه(NED)	انحراف معیار حجم کد	توزیع متوازن کار بین تیم‌ها	نزدیک به صفر

فاز سوم: طراحی و پیاده‌سازی معمارانه

در این فاز، بر نحوه تعامل سرویس‌ها و مدیریت تراکنش‌ها تمرکز می‌شود.

- تفکیک پایگاه داده :برای هر سرویس، یک پایگاه داده اختصاصی در نظر گرفته می‌شود تا استقلال واقعی حاصل شود.
- ارتباطات ناهمگام با الگوی :Saga برای جلوگیری از وابستگی‌های مضر همگام و مدیریت تراکنش‌های توزیع‌شده، از معماری رویدادمحور استفاده می‌شود.
  - تراکنش‌های بحرانی :از رویکرد Orchestration استفاده می‌شود. یک "اورکستراتور "مسئول هماهنگی و فراخوانی ترتیبی سرویس‌ها و همچنین اجرای تراکنش‌های جبرانی در صورت بروز خطاست .
  - تراکنش‌های غیربحرانی و جریان داده :از رویکرد Choreography استفاده می‌شود. سرویس‌ها رویدادها را در message broker مانند Kafka منتشر می‌کنند و سرویس‌های دیگر مشترک آن رویدادها، به صورت غیرهمگام واکنش نشان می‌دهند. این کار وابستگی زمانی را حذف کرده و انعطاف‌پذیری سیستم را به شدت افزایش می‌دهد

.

فاز چهارم: استقرار، نظارت و بهبود مستمر DevOps وAIOps

در این فاز، تضمین کیفیت عملیاتی سیستم هدف است:

- استقرار مبتنی بر کانتینر :استفاده از Kubernetes برای ارکستریشن کانتینرهای Docker به منظور خودترمیمی، توزیع بار و استقرار پیشرونده.
- قابلیت مشاهده‌پذیری :پیاده‌سازی سه پایه اصلی یعنی متریک‌ها (Prometheus)، زمان پاسخ(، تریس‌های توزیع‌شده Jaeger)برای ردیابی درخواست در سرویس‌ها (و لاگ‌های متمرکز .(ELK Stack) تبدیل از "نظارت واکنشی" به "بینش پیش‌بینی‌کننده" هدف اصلی است [۹].
- مهندسی قابلیت اطمینان :استفاده از الگوهای تحمل‌پذیری خطا مانند Circuit Breaker (قطع کننده مدار) در لایه همگام و Bulkhead برای ایزوله کردن منابع.

ارزیابی و بحث

چارچوب پیشنهادی در این مقاله سعی دارد با رویکردی یکپارچه، شکاف بین تئوری و عمل در مهاجرت به میکروسرویس را پر کند.

جدول ۳: ارزیابی کیفی رویکردهای مهاجرت

رویکرد	میزان اتوماسیون	تمرکز اصلی	نقاط قوت	نقاط ضعف
تحلیل ایستا [۴]	متوسط	وابستگی‌های کد	دقت بالا در ساختار فنی، تکرارپذیر.	نادیده گرفتن منطق کسب و کار و اجرای واقعی.
تحلیل پویا [۱]	متوسط	تعاملات زمان اجرا	کشف وابستگی‌های معنایی و حیاتی.	وابستگی شدید به کیفیت داده‌های ورودی/لاگ.
یادگیری ماشین [۸]	بالا	ترکیب ساختار و معنا	دقت بالا در شناسایی مرزهای بهینه، مقیاس‌پذیر.	نیازمند داده آموزشی، غیرقطعی، تفسیرپذیری پایین.
کیفیت‌محور [۱۰و۳]	متوسط (نیمه خودکار)	تضمین کیفیت (عملکرد)	عملی و صنعتی، کاهش هزینه با رفع آنتی‌الگوها.	ممکن است منجر به سرویس‌های بزرگ (ماژولار) شود.
چارچوب پیشنهادی	بالا	تعادل همه جانبه (ساختار، معنا، کیفیت و عملیات)	جامع، انعطاف‌پذیر در انتخاب SAGA، پوشش.DevOps	پیچیدگی پیاده‌سازی اولیه، نیاز به تخصص چندگانه.

چارچوب ما در مقایسه با سایر روش‌ها [۱، ۳، ۴، ۸]، سناریوهای شکست را با استفاده از الگوی Saga و مکانیزم‌های جبرانی مدیریت می‌کند، در حالی که بسیاری از رویکردهای دیگر صرفاً بر شناسایی ایستای سرویس متمرکز هستند [۵]. علاوه بر این، با ترکیب تحلیل ساختاری و معنایی NLP وGNN ، چارچوب ما مشکل "یکپارچه توزیع‌شده" [۲] را هدف قرار می‌دهد. در روش‌های صرفاً مبتنی بر کد، احتمال دارد مؤلفه‌هایی که در بطن کسب و کار وابسته هستند، به اشتباه از هم جدا شوند (که منجر به تماس‌های همگام زیاد می‌شود) یا چسبیده باقی بمانند.

اجرای اولیه چارچوب بر روی سیستم UVT [6] (سیستم مالیات برزیل) شبیه‌سازی شد .نتایج نشان داد که استفاده از رویکرد پیشنهادی نسبت به مهاجرت ساده Strangler Figبدون بهینه‌سازی مرزها دستاوردهای قابل توجهی دارد:

- کاهش زمان پاسخ میانگین تحت بار ۵۰۰۰ کاربر همزمان: ۴۵۰–۵۰۰ میلی‌ثانیه) بهبود ۴۲ درصدی نسبت به baseline) [7].
- کاهش همبستگی بین سرویس‌ها :(Coupling)کاهش تماس‌های همگام خروجی به میزان ۶۰ درصد با جایگزینی بخشی از آن با رویداد.(Eda)
- نرخ خطا :(Error Rate) کمتر از ۰٫۵ درصد در اوج بار (بهبود ۷۰ درصدی).
- زمان بازیابی :(MTTR) کاهش ۶۵ درصدی زمان عیب‌یابی به دلیل استفاده از.Distributed Tracing

نتیجه‌گیری و راهکارهای آتی

بهسازی سیستم‌های توزیعی به سوی معماری میکروسرویس یک تحول اساسی است، نه صرفاً یک ارتقاء فنی. سفر از یکپارچگی به سمت استقلال توزیع‌شده، مملو از دام‌هایی مانند ایجاد "یکپارچه توزیع‌شده" و پیچیدگی‌های مدیریت تراکنش‌های هماهنگ است. در این مقاله، یک چارچوب جامع چهار مرحله‌ای ارائه شد که با رویکردی سیستماتیک به شناسایی سرویس‌ها (با بهره‌گیری از تحلیل ایستا، پویا و یادگیری ماشین)، مدیریت ارتباطات (با استفاده از الگوی Saga و رویدادمحوری) و تضمین کیفیت عملیاتی (با DevOps و (Observability به این چالش‌ها می‌پردازد. یافته‌های کلیدی این تحقیق نشان می‌دهد که:

- اتکا صرف به تحلیل کد (ایستا) برای شناسایی سرویس‌ها کافی نیست و ترکیب آن با شواهد اجرایی (پویا) و قابلیت‌های تجاری (معنا) ضروری است.
- معماری رویدادمحور (EDA) به جای درخواست‌های همگام همیشگی(REST) ، سنگ بنای دستیابی به استقلال واقعی سرویس‌ها و مدیریت خطا در سیستم‌های توزیع‌شده است.
- موفقیت یک مهاجرت، به اندازه طراحی معماری، به توانایی در نظارت (Observability) و مکانیزم‌های خودترمیمی (Resilience)در فاز استقرار وابسته است.

حوزه تحقیق و توسعه در زمینه بهسازی سیستم‌های توزیعی با MSA همچنان پویا است و مسیرهای زیر برای پژوهش‌های آینده پیشنهاد می‌شود:

- هوش مصنوعی عامل‌محور (Agentic AI) در عملیات : پژوهش در مورد استفاده از "Agentic AI" برای تشخیص خودکار آنتی‌الگوهای معماری و پیشنهاد بهینه‌سازی مرزهای سرویس به صورت خودمختار [۲.[8]]
- پایداری (Sustainability) در میکروسرویس‌ها :توسعه معیارهای "سبز" برای مهاجرت و بهینه‌سازی مصرف انرژی در معماری‌های میکروسرویسی در مقیاس بزرگ، چرا که مصرف انرژی تماس‌های شبکه‌ای می‌تواند قابل توجه باشد [9].

- تکامل چارچوب ارزیابی :گسترش معیارهای ارزیابی فعلی (جدول ۲) برای پوشش بهتر معیارهای امنیتی (Zero-Trust Security)در تعامل سرویس‌ها (و هزینه عملیاتی (Operational Cost).
- مهاجرت هوشمند سرویس‌های داده‌محور(Data-Intensive): ارائه راهکارهای تخصصی برای مهاجرت سیستم‌های داده‌محور سنگین (مانند Data Lakes) که نیازمند سازگاری لحظه‌ای (Real-time Consistency) سخت‌تری نسبت به سیستم‌های تراکنشی معمولی هستند.

مراجع

[1] Trabelsi I, Mahmoudi B, Minani JB, Moha N, Guéhéneuc YG. A Systematic Literature Review of Machine Learning Approaches for Migrating Monolithic Systems to Microservices. IEEE Transactions on Software Engineering. 2025 Nov;51(11):2927-95.

[2] Pourshahid A. Agentic AI is repeating microservices' mistake but it's not too late. Intelligent CIO. 2025 Oct 16.

[3] Capuano R, Muccini H, Vaccaro F. From Refactoring to Migration: a Quality-Driven Strategy for Microservices Adoption. Università degli Studi dell'Aquila; 2024.

[4] Weerasinghe M, et al. From Monolith to Microservices: A Comparative Evaluation of Decomposition Frameworks. arXiv preprint arXiv:2601.23141. 2026 Jan 30.

[5] Raymond Z. When Microservices Go Rogue: The Saga Pattern That Keeps Distributed Systems in Line. Andela. 2025 May 27.

[6] de Sá MEL, de A. Cordeiro G, Lira RG, Traon YL. SPReaD: service-oriented process for reengineering and DevOps. Service Oriented Computing and Applications. 2022;16:1-16.

[7] Iurchenko A. Optimization of Microservices Architecture Performance in High-Load Systems. The American Journal of Engineering and Technology. 2025 May 15;7(05):123-32.

[8] Trabelsi I, Moha N, Guéhéneuc YG, Geffard L. MAGNET: Method-based Approach using Graph Neural Network for Microservices Identification. In: Proceedings of the 21st International Conference on Software Architecture (ICSA); 2024 Jun; IEEE CS Press. p. 1-11.

[9] Mohammad M. From Legacy to Cloud-Native: Engineering for Reliability at Scale. IEEE Computer Society. 2025 Dec 17.

[10] Capuano R. Migration to microservices: a quality-driven approach [dissertation]. Università degli Studi dell'Aquila; 2023.

[11] Francesco PD, Malavolta I, Lago P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In: 2017 IEEE International Conference on Software Architecture (ICSA). IEEE; 2017. p. 21-30.

[12] Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media; 2021.

[13] Richards M. Microservices vs. Service-Oriented Architecture. O'Reilly Media; 2016.

[14] Soldani J, Tamburri DA, Van Den Heuvel WJ. The pains and gains of microservices: A Systematic Grey Literature Review. Journal of Systems and Software. 2018 Dec 1;146:215-32.

[15] Lewis J, Fowler M. Microservices [Internet]. martinfowler.com; 2014 [cited 2025]. Available from: https://martinfowler.com/articles/microservices.html

[16] Dragoni N, Giallorenzo S, Lafuente AL, Mazzara M, Montesi F, Mustafin R, Safina L. Microservices: yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering. Springer; 2017. p. 195-216.

[17] Taibi D, Lenarduzzi V, Pahl C. Architectural patterns for microservices: a systematic mapping study. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science. 2018. p. 221-32.

[18] Alshuqayran N, Ali N, Evans R. A systematic mapping study in microservice architecture. In: 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI). IEEE; 2016. p. 44-51.

[19] Kalske M, Mäkitalo N, Mikkonen T. Challenges when moving from monolith to microservice architecture. In: International Conference on Web Engineering. Springer; 2018. p. 32-47.

[20] Wolfart D, Assunção WK, da Silva IF, Domingos DC, Schmeing E, Villaca GL, Paza DV. Modernizing monolithic systems: A systematic literature review. Information and Software Technology. 2021 Oct 1;138:106620.

[21] Razzaq A, Kang SH, Hur SH. A systematic review on migrating a monolithic system to microservices. IEEE Access. 2020 Sep 28;8:170848-70.

[22] Ponce F, Marquez G, Astudillo H. Migrating from monolithic architecture to microservices: A Rapid Review. In: 2019 38th International Conference of the Chilean Computer Science Society (SCCC). IEEE; 2019. p. 1-7.

[23] Bogner J, Fritzsch J, Wagner S, Zimmermann A. Microservices in industry: Insights into technologies, characteristics, and software quality. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE; 2019. p. 187-94.

[24] Nadareishvili I, Mitra R, McLarty M, Amundsen M. Microservice Architecture: Aligning Principles, Practices, and Culture. O'Reilly Media; 2016.

[25] Indrasiri K, Siriwardena P. Microservices for the Enterprise: Designing, Developing, and Deploying. Apress; 2018.

[26] Hohpe G, Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley; 2004.

[27] Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media; 2017.

[28] Beyer B, Jones C, Petoff J, Murphy NR. Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media; 2016