نشریه علمی «علوم و فناوری‌های پدافند نوین»
سال دهم، شماره ۳، پاییز ۱۳۹۸؛ ص ۳۳۸-۳۲۳

سامانه تصمیم‌یار برای مهاجرت برنامه‌های کاربردی محلی به ابر: بهبود دسترس‌پذیری سامانه و محافظت در برابر بلایا

امید بوشهریان[۱*]، سید یحیی نبوی[۲]
۱- دانشیار ۲- دانشجوی دکتری، دانشگاه صنعتی شیراز
(دریافت: ۹۷/۴/۰۳، پذیرش: ۹۷/۱۱/۲۴)

چکیده

استفاده از رایانش ابری در کاهش هزینه‌های عملیاتی برنامه‌های کاربردی مؤثر بوده و در کنار آن افزایش مقیاس‌پذیری، انعطاف‌پذیری، در دسترس بودن و قابلیت اعتماد را نیز به همراه دارد. از همین رو سازمان‌ها برای افزایش دسترس‌پذیری و محافظت از سامانه‌های خود در برابر تهدیدهای مختلف که ارائه خدمات مورد انتظار را با چالش مواجه می‌سازند، به دنبال مهاجرت برنامه‌های کاربردی خود به محیط ابر هستند. فرآیند مهاجرت به ابر، با توجه به پیچیدگی برنامه‌های کاربردی و محیط پویای سازمان‌ها، فرآیندی پیچیده و زمان‌بر است. باوجود تحقیقات بسیار در این رابطه تاکنون مدل مهاجرت مناسبی بر اساس الگوهای شناخته‌شده ارائه نشده است. در این مقاله جبر فرآیند حالت متناهی (FSP) به‌عنوان پایه‌ای برای ایجاد یک سامانه پشتیبان مهاجرت گام‌به‌گام به ابر مورد استفاده قرار گرفته است. این سامانه با استفاده از مشخصات برنامه‌های کاربردی و سازمان به‌صورت خودکار ساخته می‌شود. با توجه به این واقعیت که رویکرد گام‌به‌گام برای محیط‌های پویا مناسب‌تر است، مدل مهاجرت گام‌به‌گام در مقایسه با مدل‌های بهینه‌سازی موجود که برای یافتن بهترین معماری برای استقرار مؤلفه‌های برنامه کاربردی بر روی سرویس‌های ابری مورد استفاده قرار می‌گیرند، عملکرد بهتری دارد درواقع مهم‌ترین مزیت آن در مقایسه با روش‌های قبلی پشتیبانی از خودکار سازی طرح مهاجرت با امکان اصلاح مسیر مهاجرت بر اساس تغییر اهداف در محیط‌های سازمانی پویا است.

کلیدواژه‌ها: فرآیند حالت متناهی، مهاجرت به ابر، محافظت در برابر بلایا

Decision Support System for Migrating Legacy Applications to the Cloud: Improving System Availability and Protecting Against Disasters

O. Bushehrian[*], S. Y. Nabavi
Shiraz University of Technology
(Received: 24/06/2018; Accepted: 13/02/2019)

Abstract

Using cloud computing helps enterprises to reduce their operational costs as well as to improve the scalability, availability, and reliability of their services. To improve system availability and protecting against disasters, enterprises have to decide how to migrate their on-premise applications to the cloud. Migrating of a legacy application to the cloud is a very complicated and time-consuming process, due to the complexity of applications, the dynamic environment of the enterprises and the variety of available cloud services. Despite many types of research in this context, a formal migration model based on known patterns has not been presented yet. In this paper, the Finite State Process (FSP) algebra is applied as a formal basis by which a step by step migration support system can be built automatically from the known application and cloud profiles. The proposed step by step migration model is superior to the current optimization methods that search the optimal deployment of application components to cloud services due to the fact that a step by step approach is more appropriate for dynamic environments. In fact, the main advantage of this method over previous methods is that it supports the automation of the migration plan with the possibility of modifying the migration path based on changing objectives in dynamic enterprise environments.

Keywords: Finite State Process, Cloud Migration, Protecting Against Disasters.

[*]Corresponding Author E-mail: Bushehrian@sutech.ac.ir

۱. مقدمه

امروزه موضوع محافظت از داده‌ها و سرویس‌ها در برابر سوانح طبیعی و غیرطبیعی ازجمله موضوعات مهم در مباحث امنیتی است. به‌طوری‌که محافظت از دارایی‌های اطلاعاتی و ارائه خدمات در مواقع بروز حوادث و بلایا ازجمله راهبردهای مهم سازمان‌ها و حتی دولت‌ها است. در همین رابطه میزان دسترسی‌پذیری و قابلیت اعتماد سامانه‌ها یکی از معیارهای مهم در انتخاب و استفاده از آن‌ها در مراکز حساس و مهم است[۱]. در همین رابطه رایانش ابری به سازمان‌ها برای بهبود مقیاس‌پذیری، انعطاف‌پذیری، قابلیت دسترس‌پذیری و اطمینان سامانه‌ها و برنامه‌های کاربردی کمک می‌کند. ابر یک فرصت مناسب برای سازمان‌هایی فراهم می‌سازد که می‌خواهند با مهاجرت برنامه‌های کاربردی قدیمی خود بر روی ابر از مزایای آن استفاده کنند. مهاجرت یک برنامه کاربردی بر روی ابر یک فرآیند ساده نیست و با توجه به الزامات و نیازمندی‌های سازمان‌ها، ساختار برنامه‌های کاربردی و تنوع سرویس‌های ابری ارائه شده توسط ارائه‌دهندگان خدمات ابری با چالش‌های بسیاری روبرو است[۴-۲]. همچنین از آنجائی که در مهاجرت به ابر گاه معیارهای متعدد و متناقضی ازجمله هزینه، کیفیت و نوع سرویس و ... را باید در نظر گرفت، این فرآیند پیچیده و زمان‌بر است.

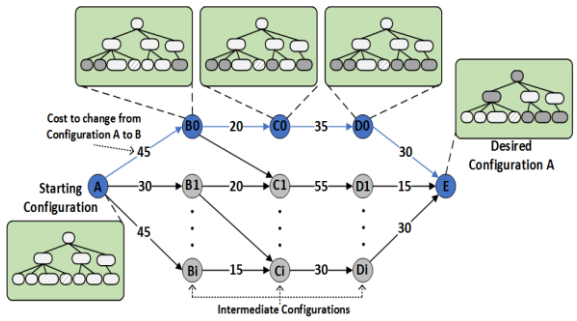
اگرچه بازه وسیعی از پژوهش‌ها در این خصوص توسط محققان انجام شده است ولی تاکنون مدل مناسبی جهت فرموله سازی طرح مهاجرت و فرآیند انتخاب سرویس ابری ارائه نشده است. چنین مدلی طراحی فرآیندی خودکار برای سهولت و ساده‌سازی مهاجرت برنامه‌های کاربردی محلی به ابر را ساده می‌کند. در این رابطه تعدادی از روش‌های موجود دستورالعمل یا راهنماهایی کلی برای مهاجرت برنامه کاربردی به ابر بر اساس تجربیات و بهترین روش ممکن را پیشنهاد نموده‌اند [۵-۳]. تعدادی دیگر معیارهایی را برای انتخاب سرویس‌های ابری مناسب برای جایگزینی مؤلفه‌های برنامه کاربردی بر اساس معیارهای دلخواه، الگوریتم‌های بهینه‌سازی چند معیاره یا مدل‌های ریاضی در نظر گرفته‌اند [۸-۶]. این در حالی است که بیشتر روش‌های پیشنهادی تنها بر استفاده از یک نوع سرویس ابری (بیشتر نرم‌افزار و زیرساخت به‌عنوان سرویس نسبت به پلتفرم به‌عنوان سرویس) تمرکز دارند. به‌طور نمونه روش‌های ارائه شده بر انتخاب ارائه‌دهندگان خدمات ابر در بخش زیرساخت به‌عنوان سرویس تمرکز نموده‌اند. این رویکردها اغلب غیرخودکار، غیرقطعی، دارای فرآیند پیچیده و نیازمند داشتن دانش فنی بالا در حین ارائه طرح مهاجرت است [۱۱-۹].

برای حل مسئله مهاجرت سامانه‌های چند مؤلفه‌ای به ابر چندین رویکرد وجود دارد. اول اینکه در ابتدا بر اساس محدودیت‌های موجود مانند (هزینه، رعایت ترتیب انتقال و ...) با استفاده از یک روش بهینه‌سازی پیکربندی بهینه نهایی تولید گردد. اشکال این روش آن است که اگرچه توسعه‌دهندگان پیکربندی بهینه نهایی را می‌دانند ولی مسیر و گام‌های رسیدن به این پیکربندی مشخص نیست. برای رفع این مشکل می‌توان از روش گام‌به‌گام استفاده نمود. در این رویکرد با توجه به محدودیت‌های موجود (مؤلفه‌ای که باید قبل از مؤلفه‌های دیگر منتقل شود و یا مؤلفه‌های خدمتگزاری که به سایرین خدمت ارائه می‌دهند)، گام‌های میانی توسط یک روش بهینه‌سازی مشخص می‌گردد و این کار تا رسیدن به پیکربندی نهایی هدف ادامه می‌یابد. به‌عبارت‌دیگر در هر گام برای یافتن مؤلفه یا مؤلفه‌های نامزد برای مهاجرت، الگوریتم بهینه‌سازی اجرا می‌شود. نکته قابل‌توجه این است که در این رویکرد کل مسیر مهاجرت در ابتدا ایجاد می‌گردد و پیکربندی‌های میانی که باید از آن‌ها عبور کنیم از ابتدا مشخص است. حال سؤال این است که آیا در عمل نیز می‌توان با طی مسیر مذکور به پیکربندی نهایی رسید. پاسخ این سؤال همواره مثبت نیست زیرا ممکن است مشکلاتی در طول عملیات مهاجرت مؤلفه‌ها به ابر به وجود آید (مانند تغییر نیاز و اولویت مشتریان، تغییر اهداف سازمان، فضای کسب‌وکار، قطع همکاری گروه توسعه و...) که موجب تغییر مسیر مهاجرت شده و درنهایت امکان رسیدن به پیکربندی نهایی از پیش تعیین شده، وجود نداشته باشد. رویکرد سوم استفاده از روش گام‌به‌گام به همراه استفاده از تابع هدف است. ورودی تابع هدف مقادیر شاخص‌های اندازه‌گیری شده مؤثر در اهداف سازمان و خروجی آن مؤلفه‌های منتخب و روش مهاجرت آن‌ها است. بدین ترتیب که در هر گام با توجه به مقادیر شاخص‌های اندازه‌گیری شده در منظرهای مختلف مانند مسائل مالی، رضایت مشتریان، قابلیت اطمینان و در دسترس بودن و سایر محدودیت که در رویکردهای قبلی مورد نظر بود، مؤلفه‌های منتخب و روش مهاجرت آن‌ها مشخص می‌گردند. پس از اجرای هر گام برای گام بعدی مجدداً مقدار تابع هدف محاسبه شده و پیکربندی میانی برای گام بعدی مشخص می‌گردند. در این رویکرد مسیر و پیکربندی‌های میانی از ابتدا مشخص نیست و بر اساس تغییرات شاخصه‌ای سازمان در طول زمان تغییر می‌کند.

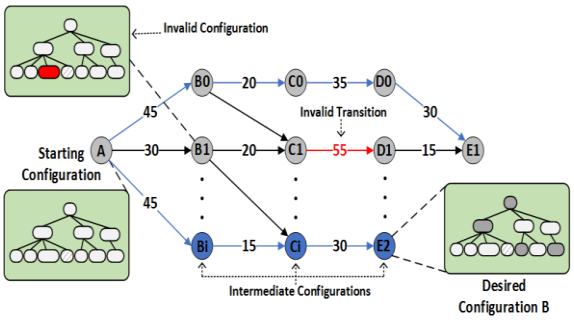
یک مسئله مهاجرت چندمرحله‌ای شامل انتقال از یک پیکربندی مبدأ با استفاده از یک سری پیکربندی‌های میانی به پیکربندی پایانی است که با مجموعه مورد نظر از الزامات مطابقت دارد. فضای راه‌حل برای تولید مجموعه‌ای از پیکربندی‌های مرتبط برای رسیدن به حالت پایانی مطلوب را می‌توان به‌صورت یک گراف جهت‌دار نشان داد (شکل ۱).

هر مجموعه از نقاط مرتبط نشان‌دهنده پیکربندی‌های بالقوه یک مدل ویژگی در هر مرحله است. برای مثال، پیکربندی‌های B_i ... B0 نشان‌دهنده پیکربندی میانی است که با یک گام از پیکربندی شروع قابل‌دسترسی هستند. در این بخش، از این گراف

برای فرموله‌سازی فضای راه‌حل‌های مسئله استفاده می‌کنیم. این فرموله‌سازی برای نمایش مسئله یافتن راه‌حل‌های معتبر استفاده می‌شود. این مسئله با چالش‌هایی همراه است ازجمله پیچیدگی گراف، محدودیت‌های پیکربندی در گره و محدودیت‌ها در تغییر پیکربندی‌ها در گذار از یک پیکربندی به پیکربندی همسایه است.



(الف)

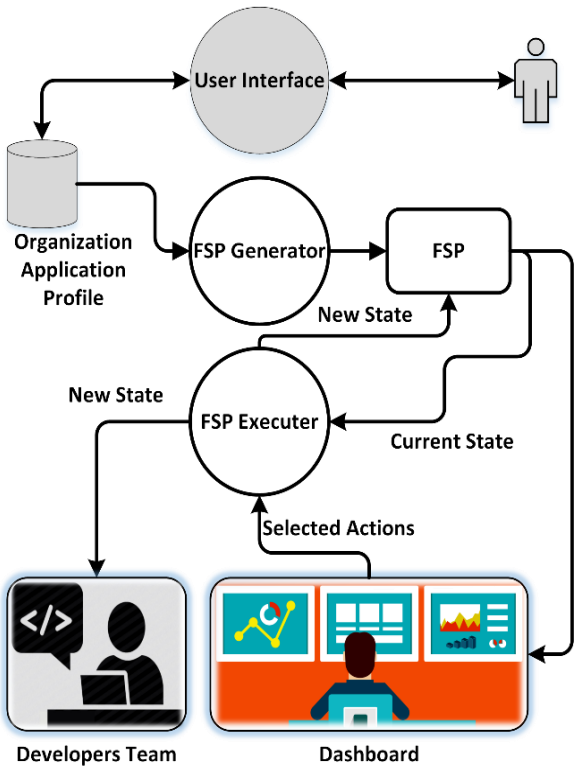


(ب)

شکل ۱. مهاجرت چندمرحله‌ای الف) نمایش مسیر رسیدن از نقطه شروع به پیکربندی پایانی و نقاط میانی ب) نمایش گذر نامعتبر (یال گره C1 و D1) و پیکربندی نهایی جایگزین (نقطه E2)

در این مقاله، روش نوینی بر اساس جبر فرآیند حالت متناهی[1] [۱۳] به‌منظور طراحی طرح جامع، قطعی و قابل‌گسترش مهاجرت به ابر برای غلبه بر چالش‌های فوق، ارائه شده است. فرآیند حالت متناهی ماشین انتزاعی است که برای مدل نمودن رفتار سامانه‌های توزیع‌شده و هم‌زمان استفاده می‌شود. به‌این‌ترتیب، مهاجرت هر مؤلفه از برنامه کاربردی می‌تواند توسط یک فرآیند حالت متناهی مدل شود. این فرآیند شامل انجام دنباله‌ای قابل تکرار از اقدامات مشخص در جهت مهاجرت آن مؤلفه به ابر است. بنابراین، یک طرح مهاجرت را می‌توان مجموعه‌ای از فرآیندهای هم‌زمان برای مهاجرت بخشی از برنامه کاربردی به ابر در نظر گرفت که برخی از اقدامات آن‌ها با یکدیگر هماهنگ شده‌اند. مهم‌ترین مزیت استفاده از فرآیند حالت متناهی برای مدل نمودن طرح مهاجرت، ایجاد یک طرح مهاجرت گام‌به‌گام و سازگار با شرایط در حال تغییر سازمان است. از این طرح در مقابل با طرح ایستا، یک مرحله‌ای و طولانی مدت- که به‌طور کامل هم عملی نیست-با توجه به شرایط پویای سازمان بهتر عمل می‌کند.

برای پیاده‌سازی سامانه تصمیم‌یار مورد نظر در این پژوهش، ابتدا الگوریتمی برای تولید کد FSP طرح مهاجرت با توجه به مشخصات برنامه کاربردی و سرویس‌های ابری ارائه شده است. این الگوریتم در قسمت تولیدکننده FSP[2] قرار می‌گیرد و وظیفه آن تولید کد FSP بر اساس مشخصات برنامه کاربردی و سرویس‌های ابری[3] است. سپس برای اجرا طرح مهاجرت، اجراکننده کد FSP[4] نیز طراحی و تولید شده است. وظیفه این اجراکننده، اجرای کد FSP تولید شده در مرحله قبل است. اجرای این کد شامل پیشنهاد مجموعه اقدامات قابل‌اجرا در هر مرحله به مدیر و همچنین اجرای اقدامات منتخب مدیر در جهت مهاجرت به ابر است. مدل ارائه شده در این مقاله در شکل (۲) نشان داده شده است.



شکل ۲. مدل ارائه شده برای سامانه تصمیم‌یار در این مقاله

در ادامه این مقاله بعد از بخش مقدماتی، بخش دوم به جمع‌بندی مطالعات مرتبط با توجه به مدل‌های مبتنی بر الگو اختصاص دارد. بخش سوم با مروری بر الگوهای مهاجرت ارائه شده به تعریف جبر فرآیند حالت متناهی می‌پردازد. همچنین در این بخش ارتباط میان طرح مهاجرت و فرآیند حالت متناهی شرح داده می‌شود. در بخش چهارم الگوریتمی برای تولید خودکار

[1] Finite State Process (FSP) Algebra

[2] FSP Generator
[3] Application and Cloud Profiles
[4] FSP Executer

طرح مهاجرت بر اساس مدل فرآیند حالت متناهی معرفی می‌گردد. در بخش پنجم الگوریتم پیشنهادی به‌صورت بررسی موردی بر روی برنامه کاربردی نمونه اجرا و نتایج نمایش داده می‌شود. در بخش ششم با استفاده از الگوریتم ارائه شده، سامانه‌ای برای اجرای طرح مهاجرت معرفی شده است و درنهایت، بخش هفتم به نتیجه‌گیری و پیشنهاد کارهای آینده اختصاص دارد.

۲. پیشینه تحقیق

همان‌گونه که در بخش‌های قبلی به آن اشاره شده مهاجرت به ابر فرآیندی پیچیده است و این حقیقت که هنگام انتخاب سرویس‌های ابری، مهندسین و طراحان باید یک مجموعه شرایط و معیارهای ناهمگون با وابستگی‌های پیچیده را در نظر بگیرند، امری است که انجام آن به‌صورت دستی ناممکن به نظر می‌رسد و این موضوع یک چالش بزرگ است. بااین‌وجود افزایش محبوبیت و مزایای استفاده از رایانش ابری محققان را به تلاش‌های زیاد برای ارائه راه‌حل و ابزاری مناسب جهت تسهیل مهاجرت به ابر ترغیب نموده است [۴]. پیچیدگی تصمیم‌گیری برای مهاجرت به ابر در کنار تکامل و محبوبیت محاسبات ابری منجر به توجه بسیاری از مدیران صنایع و مراکز تحقیقاتی به طراحی سامانه‌های پشتیبان تصمیم‌گیری مهاجرت به ابر شده است. ارائه‌دهندگان خدمات ابر و سازمان‌های مشاوره‌ای فناوری اطلاعات نیز تلاش‌هایی را برای پرداختن به این تقاضا با ارائه دستورالعمل‌ها و ابزارهای ارزیابی متعدد انجام داده‌اند بااین‌حال، اقدامات صورت‌گرفته بیشتر به‌منظور نیل به اهداف بازاریابی بوده و در دسترس عموم قرار ندارند زیرا اغلب آن‌ها در قالب قراردادهای مشاوره‌ای و با استفاده از فناوری ویژه انجام می‌شوند [۱۴، ۵، ۳]. در مقابل نیز طیف گسترده‌ای از روش‌های مهاجرت به ابر در مراکز تحقیقاتی مطرح شده‌اند. بررسی‌های انجام شده با هدف شناسایی، طبقه‌بندی و مقایسه تحقیقات موجود در جهت مهاجرت سامانه‌های محلی موجود به سمت سرویس‌های مبتنی بر ابر نشان‌دهنده فقدان روش‌های قابل تکرار و اثبات پذیر است. درواقع این موضوع به‌عنوان یکی از دلایل اصلی عدم بلوغ کامل حوزه مهاجرت به ابر نیز عنوان شده است[۲].

از زمان ظهور پارادایم محاسبات ابری، طیف گسترده‌ای از راهبردها و رویکردها به‌منظور کمک به مدیران و توسعه‌دهندگان برای مهاجرت برنامه‌های کاربردی قدمی خود به ابر ارائه شده است. برخی از پژوهش‌ها با انجام مطالعه بر روی عملیات مهاجرت واقعی و عملی سامانه‌های نرم‌افزاری به ابر، تجربیات برتر و آموزه‌های به‌دست‌آمده را گزارش نموده‌اند. آن‌ها دستورالعمل‌های تجربی را فراهم آورده‌اند که برای مهاجرت برنامه‌های کاربردی مفید است [۲، ۳، ۴]. خواجه‌حسینی و همکاران [۲] یک مطالعه موردی از صنعت نفت و گاز را شرح می‌داده‌اند که در آن مرکز داده‌ها برون‌سپاری شده به آمازون مهاجرت داده شده است. باوجود مزایای بسیاری مانند صرفه‌جویی ۳۷٪ در هزینه‌های تعمیر و نگهداری و حذف ۲۱٪ از تماس‌های پشتیبانی، برخی از خطرات فنی-اجتماعی[1] مانند تضعیف کیفیت خدمات و توجه به مشتری مشاهده شده است. به‌عنوان نمونه دیگر تجربیات به‌دست‌آمده از مهاجرت یک نرم‌افزار متن باز بنام Hackystat به ابر، گزارش شده است [۱۵]. مؤلفان قبل از فرآیند مهاجرت، مطالعاتی را برای به‌روزرسانی معماری نرم‌افزار برای انطباق با نیازهای SaaS و IaaS انجام داده‌اند. چارچ و همکاران [۱۶] ، مهاجرت یک سامانه صنعتی حیاتی را بررسی کرده‌اند. این بررسی شامل استفاده از ترکیب روش تغییر میزبان و آزمون عملکرد برای مهاجرت یک برنامه کنترل نظارتی بر روی ابر تحقیقاتی NeCTAR و جمع‌آوری اطلاعات[2] مربوط به آن است. آن‌ها بر روی عوامل مؤثر بر عملکرد سامانه مذکور بر روی ابر مانند نوع پروتکل جمع‌آوری داده (رویداد محور یا سرکشی) و تکثیر سرورهای راه دور متمرکز شده است.

قلمرو مهاجرت سامانه‌های نرم‌افزاری به ابر و فعالیت‌های مربوط به آن مانند: (۱) آشنایی با سامانه نرم‌افزاری و ابر هدف، (۲) تغییرات کد و (۳) ارزیابی نهایی، شناسایی شده است. آن‌ها از این فعالیت‌ها برای مهاجرت نرم‌افزار Pet Shop. Net به ویندوز Azureو پایگاه داده SQL Azure استفاده کرده‌اند. طبقه‌بندی فعالیت‌های مهاجرت و عوامل مؤثر بر هزینه این کار نیز ازجمله مشخص شده در تحقیقات است [۱۷]. در پژوهشی دیگر یک فرایند گام‌به‌گام کلی و اصولی برای پشتیبانی از مهاجرت برنامه‌های کاربردی به ابر بر اساس پروفایل سازمان، برنامه کاربردی و ابر ارائه شده است. این رویکرد از توسعه‌دهندگان پشتیبانی می‌کند، بااین‌وجود مهاجرت به ابر نیازمند یک مکانیسم تصمیم‌گیری در سطح مدیریت است. ضعف دیگر این روش عدم ارائه ابزار خودکار به‌منظور انتخاب بهترین ارائه‌دهنده ابر است [۱۸].

تعدادی از راهبردهای مهاجرت مورد استفاده و الگوهای معماری مربوط به آن‌ها در پژوهشی بنیادین ارائه شده است [۱۹]. دو نوع اصلی از الگوهای مهاجرت Cloudification و Cloud Hosting در این مقاله موردبررسی قرار گرفته و زیر نوع‌های مختلف آن به همراه مثال عملی توضیح داده شده است. در مقاله‌ای دیگر تجربیات برتر برای مهاجرت یک نرم‌افزار مبتنی بر سرویس به ابر با استفاده از الگوهای قابل‌استفاده مجدد مورد مطالعه قرار گرفته است[۲۰]. محققان به این نتیجه رسیده‌اند که برنامه‌های کاربردی مبتنی بر سرویس به علت داشتن معماری با اتصال ضعیف، انتخاب‌های مناسبی برای مهاجرت به ابر هستند.

[1] Socio-Technical
[2] Eclipse Supervisory Control and Data Acquisition (SCADA)

جمشیدی و همکاران[۲۱]، مجموعه‌ای از الگوهای برای طبقه‌بندی فرآیند مهاجرت ارائه داده‌اند. الگوها می‌توانند توسط گروه توسعه نرم‌افزار برای استفاده از یک طرح مهاجرت با توجه به اهدافی مانند کاهش زمان عرضه به بازار، کاهش هزینه‌های عملیاتی، آزادسازی منابع محلی، مقیاس‌پذیری بالا و بهبود بهره‌وری عملیاتی، مورد استفاده قرار گیرد. با این حال در این روش محاسبه هزینه مسیرهای مختلف مهاجرت با توجه به اهداف سازمانی و محدودیت‌ها و الزامات برنامه کاربردی مشکل است. علاوه بر این تغییر در اهداف سازمان نیز در انتخاب طرح مهاجرت در نظر گرفته نشده است.

در برخی از آثار، مسئله مهاجرت به ابر به‌عنوان یک مسئله بهینه‌سازی برای یافتن بهترین معماری استقرار مؤلفه‌های نرم‌افزاری بر روی ابر در نظر گرفته‌اند [۲۴-۲۲]. لیمن و همکاران [۲۲] یک روش مهاجرت برای به دست آوردن توزیع مطلوب نرم‌افزارهای ترکیبی بر روی چندین ارائه‌دهنده ابر ارائه کرده است. در این پژوهش یک روش برای تقسیم نرم‌افزار به قطعات مجزا و سپس استقرار خودکار آن‌ها بر روی ابر، با توجه به میزان بار کاری و عملکرد مورد انتظار هر مؤلفه پیشنهاد شده است. آن‌ها این روش را با برخی از روش‌های بهینه‌سازی مانند الگوریتم کوهنوردی[۱] و تکاملی[۲]، ارزشیابی نموده‌اند. نتایج ارزیابی حاکی از عملکرد مناسب الگوریتم تپه‌نوردی با توجه به زمان اجرای است. در پژوهشی دیگر مؤلفین راه‌حل مبتنی بر مدل بهینه‌سازی خطی عدد صحیح[۳] برای مهاجرت به مسئله ابر ارائه کرده‌اند[۲۳]. هدف یافتن پاسخ با حداقل هزینه با توجه به مجموعه محدودیت‌هایی است که نیاز مشتری را تشریح می‌کنند. محققان به‌عنوان راه‌حلی دیگر روش مدل ویژگی[۴] را برای مدل‌سازی فضای پیکربندی‌های موجود یک ارائه‌دهنده خدمات ابری ارائه کرده‌اند [۲۴]. مدل ویژگی، یک مدل ساختاریافته و جمع‌وجور برای بیان مشخصات الزامات و نیازمندی‌های مشتریان و همچنین بیان مشخصات پیکربندی زیرساخت‌های ابری است. پس از مدل‌سازی، یک جستجو برای یافتن بهترین FM های ابری متناظر با FM نرم‌افزار انجام می‌شود.

هدف این مقاله ارائه یک الگوریتم برای ایجاد یک طرح مهاجرت بر مبنای زبان رسمی فرآیند حالت متناهی (FSP) با استفاده از مشخصات سازمانی و سرویس‌های ابری است. مدل FSP بر مبنای الگوها و مطابق با الزامات سازمان ایجاد می‌شود به‌طوری‌که مهاجرت هر مؤلفه در هماهنگی با سایر مؤلفه‌های مرتبط صورت می‌گیرد. مزیت کلیدی و نوآوری روش ارائه شده در این مقاله، ضابطه‌مند نمودن طرح مهاجرت به ابر با استفاده از فرآیند حالت متناهی است به‌نحوی‌که فرآیندهای ارائه‌شده صریح، جامع و قابل‌گسترش باشند.

۳. پیش‌زمینه

در خصوص مهاجرت به ابر تعاریف مختلفی ذکر شده است. برای هماهنگی و درک مشترک از مسئله مهاجرت به ابر مطالبی در این بخش ارائه شده است.

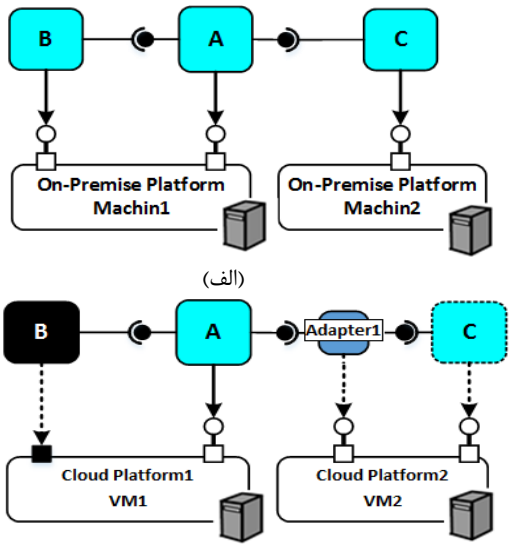
۳-۱. مهاجرت به ابر

منظور از مهاجرت به ابر در این مقاله، عبارت است از فرآیند انتقال تمام یا بخشی از منابع فناوری اطلاعات قدیمی (به معنای منابع رایانه‌ای موجود سازمان که از قبل مانده بکار می‌رود تا آن‌ها را از طراحی و به‌کارگیری سامانه‌های جدید جدا سازد) یک سازمان، ازجمله: سخت‌افزار، نرم‌افزار، داده‌های ذخیره‌شده و فرآیندهای کسب‌وکار از محل استقرار حاضر (محلی) به محیط ابر است، به‌طوری‌که آن‌ها را از راه دور توسط شرکت ارائه‌دهنده سرویس ابری مدیریت نمود. این فرآیند شامل انتقال منابع به ارائه‌دهندگان ابرهای مختلف نیز است. همچنین فرآیند مهاجرت به ابر الزاماً به معنی انتقال همه منابع به محیط ابری نیست و ممکن است منجر به حفظ برخی زیرساخت‌ها و منابع به‌صورت محلی گردد.

به‌طورمعمول انگیزه سازمان‌ها برای توجه به موضوع مهاجرت به ابر مواردی چون کسب مزیت رقابتی، بهبود بهره‌وری، چابکی کسب‌وکار و نوآوری است. مهاجرت به ابر یک تصمیم راهبردی برای سازمان به‌منظور ارتقای توسعه این اهداف از طریق بهبود مقیاس‌پذیری، انعطاف‌پذیری و زمان عرضه به بازار (زمان صرف شده از ایده یک محصول تا روانه شدن آن به بازار) است [۲۵].

شکل (۳- الف و ب) سناریویی را برای مهاجرت یک برنامه کاربردی نشان می‌دهند. در ابتدا یک برنامه کاربردی از سه مؤلفه A، B و C تشکیل شده است که در آن مؤلفه A به مؤلفه‌های B و C سرویس می‌دهد. همچنین مؤلفه A و B بر روی ماشین فیزیکی ۱ و مؤلفه C بر روی ماشین فیزیکی ۲ مستقر هستند. شکل (۳- ب) نحوه استقرار مؤلفه‌ها را بعد از مهاجرت نشان می‌دهد. مؤلفه‌های A و B بر روی ماشین مجازی ۱ بر روی ابر ۱ و مؤلفه C بر روی ماشین مجازی ۲ روی ابر ۲ مستقر شده‌اند. همان‌طور که در شکل (۳- ب) مشاهده می‌شود برخی مؤلفه‌ها برای مهاجرت نیاز به تغییر دارند (مؤلفه C) و در برخی موارد برای برقراری اتصال مجدد میان مؤلفه‌های مهاجرت یافته با یکدیگر و یا با سایر مؤلفه‌ها به مبدل نیاز است. (مبدل ۱ میان مؤلفه A و C). در بخش‌های بعدی انواع مهاجرت به شرح داده خواهد شد.

[1] Hill Climbing
[2] Evolutionary Method
[3] Integer Linear Programming (ILP)
[4] Feature Modeling



شکل ۳. مثال استفاده از روش‌های مهاجرت به ابر الف) برنامه کاربردی دارای سه مؤلفه A، B و C ب) طرح مهاجرت مربوط به برنامه کاربردی شکل الف

۳-۲. الگوهای مهاجرت

محققان در پژوهش‌های مربوط به مهاجرت به ابر الگوهای مختلفی را پیشنهاد نموده‌اند. مطابق الگوهای ارائه شده، به‌طورکلی برای انتقال یک مؤلفه از برنامه کاربردی به ابر دو روش وجود دارد [۱۹]. روش اول که میزبانی ابر نام دارد، شامل انتقال کد منبع مؤلفه از محیط محلی به محیط ابری است که ممکن است با بازنویسی کد مؤلفه نیز همراه شود. روش دوم، ابری‌سازی نام دارد که شامل جایگزینی سرویسی‌های ارائه‌شده توسط مؤلفه مورد نظر برای مهاجرت با یک یا چند سرویس ابری است. لازم به ذکر است پس از انتقال کد منبع یک سرویس محلی به ابر یا جایگزینی آن با سرویس ابری، ممکن است فراخوانی سایر سرویس‌ها به آن با روش قبلی امکان‌پذیر نباشد، در این حالت لازم است از یک یا دو مبدل برای سازگاری رابط سرویس‌ها استفاده شود و یا کد مؤلفه‌ها تغییر یابد. در ادامه انواع مختلف این دو رویکرد تشریح می‌گردد.

روش میزبانی ابر[1]: الگوهای مختلف این روش عبارت‌اند از: انقیاد مجدد، انطباق سرویس، تبدیل سرویس، جبران سرویس

- انقیاد مجدد: در این الگوی ساده یک مؤلفه برنامه کاربردی بدون هیچ‌گونه تغییری در کد منبع آن به محیط ابری منتقل می‌گردد. بعد از مهاجرت، مؤلفه مذکور می‌تواند از منابع ابری استفاده کند.
- انطباق سرویس: استفاده از این الگو زمانی است مؤلفه مهاجرت یافته (هدف) قادر به برقراری ارتباط قبلی خود با سایر مؤلفه‌های خارج از ابر- ازجمله کاربرانش- نیست. در اینجا از مؤلفه‌های مبدل در کنار مؤلفه مهاجرت یافته/خارجی برای برقراری ارتباط غیرمستقیم بین آن‌ها استفاده می‌گردد. همانند شکل (۳- ب) که در آن مؤلفه A و C توسط مبدل۱ (Adapter1) باهم در ارتباط‌اند.
- تبدیل سرویس: در این الگو توسعه‌دهندگان به‌جای استفاده از مبدل‌ها برای برقراری ارتباط میان مؤلفه هدف با مؤلفه‌های خارج از ابر، برقراری ارتباط را با تغییر کد منبع دو مؤلفه تحت تأثیر ممکن می‌سازند، مانند مؤلفه C در شکل (۳- ب).
- جبران سرویس: اگر راه‌حلی برای برقراری ارتباط میان مؤلفه هدف با سایر مؤلفه‌های وابسته در خارج از ابر با استفاده از روش‌های بالا ممکن نباشد، راه‌حل موجود تغییر مؤلفه هدف به‌طوری است که پس از مهاجرت بتواند عدم دسترسی به سرویس مؤلفه‌های خارجی را "جبران" کند. این کار به دو روش امکان‌پذیر است: اول، غیرفعال کردن قابلیت‌های موردنیاز مؤلفه هدف که از طریق سایر مؤلفه‌ها تأمین می‌گردد و دوم، اضافه نمودن قابلیت‌های موردنیاز مشابه به‌عنوان بخشی از کد اصلی مؤلفه هدف توسط توسعه‌دهندگان است.

روش ابری‌سازی[2]: این روش یکی از الگوهای جایگزینی، انطباق رابط و تبدیل رابط را شامل می‌شود.

- جایگزینی: همانند الگوی انقیاد مجدد در روش میزبانی ابر، جایگزینی یک مؤلفه با سرویس ابری ساده‌ترین شکل ابری‌سازی است. استفاده از این راه‌حل تنها زمانی امکان‌پذیر است که مؤلفه هدف و سرویس ابری نامزد دارای رابط یکسان یا به‌طور کامل سازگار باشند. به‌عنوان نمونه مؤلفه B در شکل (۳- ب) از این الگو استفاده می‌کند.
- انطباق رابط: اگر پس از جایگزینی یک مؤلفه با سرویس ابری، سایر مؤلفه‌ها مرتبط و سرویس ابری دارای رابطی ناسازگار باشند، یک راه‌حل مناسب پیکربندی مجدد مؤلفه‌های مرتبط برای دسترسی به سرویس ابری از طریق مبدل است که می‌تواند در داخل و یا خارج ابر واقع شده باشد.
- تبدیل رابط: این حالت مانند الگوی انطباق سرویس است، با این تفاوت که توسعه‌دهندگان به‌جای استفاده از مبدل، رابط سازگار موردنیاز را به‌طور مستقیم در کد منبع هر یک از مؤلفه‌های مرتبط مستقر می‌کنند.

[1] Cloud Hosting

[2] Cloudification

جدول ۱. الگوهای مهاجرت و اقدامات مربوط به مؤلفه‌های وابسته

روش مهاجرت	الگوی مهاجرت مؤلفه هدف	اقدامات مؤلفه هدف	اقدامات مؤلفه‌های وابسته (مشتری)
میزبانی ابر Cloud Hosting	انقیاد مجدد Rebinding	انتقال کد منبع مؤلفه بر روی ابر	برقراری مجدد ارتباط
	انطباق سرویس Service Adaptation	انتقال کد منبع مؤلفه بر روی ابر و استفاده از مبدل	برقراری مجدد ارتباط و استفاده از مبدل
	تبدیل سرویس Service Conversion	انتقال کد منبع مؤلفه بر روی ابر و تغییر کد منبع مؤلفه هدف	برقراری مجدد ارتباط و تغییر کد منبع مؤلفه وابسته
	جبران سرویس Compensation	انتقال کد منبع مؤلفه بر روی ابر، حذف سرویس موردنیاز مؤلفه هدف و یا تغییر کد منبع مؤلفه هدف	-
ابری‌سازی Cloudification	جایگزینی Replacement	جایگزینی مؤلفه با سرویس ابری	برقراری مجدد ارتباط
	انطباق رابط Interface Adaptation	جایگزینی مؤلفه با سرویس ابری و استفاده از مبدل	برقراری مجدد ارتباط و استفاده از مبدل
	تبدیل رابط Interface Conversion	جایگزینی مؤلفه با سرویس ابری	تغییر کد منبع مؤلفه وابسته

۳-۳. فرآیند حالت متناهی

یکی از روش‌های موردقبول برای مدل‌سازی رفتار سامانه‌های توزیع‌شده و هم‌زمان جبر فرآیند حالت محدود (FSP) است [۱۳]. یک فرآیند حالت محدود شامل دنباله‌ای از اقدامات متوالی و مرتبط است که می‌توان آن را با یک ماشین حالت محدود مدل نمود. در حقیقت هر FSP یک ماشین حالت محدود شامل چندین حالت و انتقال بین آن‌ها است. درFSP، اجرای دنباله‌ای از اقدامات اتمیک، منجر به انتقال از حالت فعلی به حالت بعدی می‌شود [۲۶] . هر اقدام ممکن است معانی مختلفی داشته باشد، برای نمونه : شروع سرویس‌دهی، درخواست یک سرویس خاص، ارسال پاسخ درخواست به مشتری، برقراری مجدد ارتباط بین دو مؤلفه، درخواست یک سرویس ابری یا جایگزینی یک مؤلفه با سرویس ابری.

۴. تولید طرح مهاجرت بر اساس فرآیند حالت محدود

یک طرح مهاجرت دنباله‌ای از اقدامات است که باید برای انتقال تمام یا بخشی از منابع فناوری اطلاعات ازجمله سخت‌افزار، نرم‌افزار، فضای ذخیره‌سازی داده‌ها از محیط محلی به محیط ابر اجرا شود. بر همین اساس یک طرح مهاجرت به‌طور رسمی به‌عنوان مجموعه‌ای از فرآیندها تعریف می‌شود که هر فرآیند نماینده مهاجرت یک مؤلفه به ابر است. در این بخش ما به دنبال روشی برای تولید خودکار کد FSP طرح مهاجرت هستیم.

بدین ترتیب در ابتدا تابع *CloudServices()* به‌عنوان یک نگاشت بین مجموعه مؤلفه‌ها (Comp) و مجموعه سرویس‌های ابری (CS) به شکل زیر تعریف می‌شود:

$$CloudServices : Comp \rightarrow CS \quad (۱)$$

با مهاجرت یک مؤلفه به ابر، این مؤلفه در مقابل هر یک از مؤلفه‌های وابسته از الگوی متفاوتی برای مهاجرت استفاده می‌کند. به‌عبارت‌دیگر هر مؤلفه دارای دو مجموعه اقدامات مختلف است یک مجموعه برای مهاجرت مؤلفه هدف و مجموعه دیگر اقداماتی که سایر مؤلفه‌های وابسته به مؤلفه هدف برای مطابقت با رابط آن و برقراری مجدد اتصال انجام می‌دهند. اقداماتی که برای مطابقت با مؤلفه هدف (مهاجرت کننده) قابل انجام است در جدول (۱) آورده شده‌اند. این اقدامات عبارت‌اند از انقیاد مجدد (تنظیم مجدد اتصال)، استفاده از مبدل و یا تغییر کد منبع مؤلفه.

برای هر جفت مؤلفه‌های C_i و C_j یک رابطه پیش‌شرطی می‌تواند تعریف شود. به این معنی که مؤلفه C_i نمی‌تواند استقرار یابد تا زمانی که مؤلفه Cj مستقر شود. تابع*PreDeploy()*، C_i را به مجموعه C_j شامل پیش‌نیازهای استقرار C_i نگاشت می‌کند. در این رابطه *Comp* مجموعه مؤلفه‌ها است:

$$PreDeploy : Comp \rightarrow 2^{Comp} \quad (۲)$$

هر مؤلفه وابسته به یک مؤلفه مهاجرت یافته، باید اتصال خود را به این مؤلفه با استفاده از یکی از اقداماتی که در جدول (۱) آمده است، مجدداً برقرار نماید. برای یافتن مؤلفه‌های وابسته از توابع *ServerComponents()* و *ClientComponents()* استفاده می‌شود. این توابع به ترتیب یک مؤلفه مشخص را به مؤلفه‌های سرویس‌دهنده به آن و یا مصرف‌کننده سرویس از آن نگاشت می‌کنند:

$$ServerComponents : Comp \rightarrow 2^{Comp} \quad (۳)$$

$$ClientComponents : Comp \rightarrow 2^{Comp} \quad (۴)$$

به‌عبارت‌دیگر خروجی تابع *ServerComponents(A)* نشان‌دهنده تمامی مؤلفه‌هایی است که به مؤلفه A سرویس می‌دهند و مؤلفه A مشتری آن‌ها محسوب می‌شود. به همین ترتیب خروجی تابع *ClientComponents(B)* شامل تمامی مؤلفه‌هایی است که از مؤلفه B سرویس می‌گیرند و مشتری مؤلفه B محسوب می‌شوند.

ساختار کلی الگوریتم پیشنهادی برای تولید FSP طرح مهاجرت در شبه کد تابع AlgorithmFSP_Migration_Plan آمده است.

```
(1)  AlgorithmFSP_Migration_Plan
                (Application APP, Cloud_Profile CSP)
(2)  begin
(3)     Initialize Comp, CloudServices, PreDeploy, ClientComponents,
        ServerComponents using APP and CSP
(4)     FSPmodel F=Create FSPmodel();
(5)     foreach Component i in Comp
(6)        F.AddComponent (i,PreDeploy[i], CloudServices[i],
                   ServerComponents[i], ClientComponents[i]);
(7)     foreach Service sCloudServices
(8)        F.AddService (s);
(9)     F.Synchronizer(Comp)
(10)    F.Compose(Comp , CloudServices)
(11) end
```

در ادامه توابع و زیربخش‌های مختلف بدنه الگوریتم توضیح داده شده است. باید توجه داشت که هر تابع درون بدنه الگوریتم بخشی از کدهای FSP را تولید می‌کند و آن‌ها را به مدل نهایی FSP می‌افزاید. تابع*Emit()* برای این منظور مورد استفاده قرار می‌گیرد[۲۶].

۴-۱. تابع AddComponent()

همان‌طور که در بالا به آن اشاره شد برای هر مؤلفه i، ما به فرآیند COMP[i] برای توصیف رفتار آن نیازمندیم. در ابتدای استقرار، هر مؤلفه باید منتظر استقرار تمام مؤلفه‌های پیش‌نیازش باشد. این انتظار توسط تابع wait برای تمامی مؤلفه‌های مذکور پیاده‌سازی شده است (خطوط ۵ تا ۸ در شبه کد تابع *AddComponent()*).

سپس یکی از روش‌های ابری‌سازی یا میزبانی ابر (فرآیندCLF[i] یا HST[i]) دنبال می‌شود و درنهایت رفتار آن به‌مانند فرآیند DEPLOYED[i] (خطوط ۱۲ تا ۱۴ در شبه کد تابع *AddComponent()*) ادامه می‌یابد. به ازای هر سرویس ابری S_j که می‌تواند به‌جای مؤلفه i (Comp[i]) استقرار یابد، یکی از فرآیندهای H[i][j] یا F[i][j] تعریف می‌شود. فرآیند اول، روش میزبانی ابر و فرآیند دوم روش ابری‌سازی را برای استقرار مؤلفه i به کار می‌گیرد. در اولین اقدام این فرآیندها برای مهاجرت، مؤلفه i باید انتقال خود را با ارسال علامت به اطلاع تمام مشتریانش برساند. برای این منظور، اقدام "signal_client[i]" از فرآیند COMP[i] با اقدام "signal_client[i]" از فرآیند CONNECT[i][j] و اقدام "start_reconnect[i][j]" از فرآیند CONNECT[i][j] با اقدام "start_reconnect[i][j]" از مؤلفه مشتری C_j هم‌زمان می‌گردد. بدین ترتیب هر مشتری خود برای برقراری اتصال مجدد با مؤلفه i تلاش می‌کند (خطوط ۱۵ تا ۲۱ در شبه کد تابع *AddComponent()*). فرآیند CONNECT[i][j] برای هر مؤلفه مشتری C_j از COMP[i] ایجاد شده است. این فرآیند در بخش بعدی توضیح داده خواهد شد. نکته بعدی این است که در صورت مهاجرت مؤلفه A وظیفه برقراری مجدد اتصال میان این مؤلفه و مؤلفه‌های سرویس‌دهنده به A بر عهده خود مؤلفه مهاجرت کننده (A) است. برای این منظور زمانی که عملیات استقرار مؤلفه A به پایان رسید. این مؤلفه با استفاده از اقدامات "post_rebind"، "post_adapt"، "post_conv" و یا "post_compensate" برای برقراری مجدد اتصالات با سرویس‌دهنده‌های خود تلاش می‌کند. این اقدامات به ازای هر سرویس‌دهنده به A باید در نظر گرفته شود (خطوط ۲۳ تا ۴۰ در شبه کد تابع *AddComponent()*).

```
(1)  AddComponent (i, PreDeploy[i], CloudServices[i],
     ServerComponents[i] , ClientComponents[i])
(2)  begin
(3)    Emit (COMP[i]=);
(4)    foreach Component C in PreDeploy[i] begin
(5)        j = C.index
(6)        St = St.Add (wait[j][i]→)
(7)    end for
(8)    if (St is null)St=nowait[i]
(9)        Emit(St→HST[i] | St→CLF[i])
(10)   end if
(11)   Emit(DEPLOYED[i] =
(12)              shutdown[i] → start_redeploy[i] →COMP[i]
(13)            | shutdown[i] → restart[i] →DEPLOYED[i]
(14)   foreach Server S in ServerComponents[i] begin
(15)     j = S.index
(16)     Emit(|start_reconnect[j][i] → CRECONNECT[j][i])
(17)     Emit(CRECONNECT[j][i]= post_rebind[j][i]→DEPLOYED[i] )
(18)     Emit(CRECONNECT[j][i]= post_adapt[j][i]→DEPLOYED[i])
(19)     Emit(CRECONNECT[j][i]= post_conv[j][i]→DEPLOYED[i])
(20)     Emit(CRECONNECT[j][i]=post_compensate[j][i]→DEPLOYED[i])
(21)   end for
(22)   foreach Server S in ServerComponents[i] begin
(23)       if (there is next server in ServerComponents[i]) begin
(24)          j = S.index
(25)          Emit(SRECONNECT[i][j]=
(26)          Emit(| post_rebind [i][j]→ SRECONNECT [i][j+1] )
(27)          Emit(| post_adapt [i][j]→ SRECONNECT [i][j+1] )
(28)          Emit(| post_conv [i][j]→ SRECONNECT [i][j+1] )
(29)          Emit(| post_compensate [i][j]→ SRECONNECT [i][j+1])
(30)       endif
(31)       else begin  /*for last server in list*/
(32)          j = S.index
(33)          Emit(SRECONNECT[i][j]=
(34)          Emit(| post_rebind [i][j]→DEPLOYED[i])
(35)          Emit(| post_adapt [i][j]→ DEPLOYED[i])
(36)          Emit(| post_conv [i][j]→ DEPLOYED[i])
(37)          Emit(| post_compensate [i][j]→DEPLOYED[i])
(38)       endelse
(39)   end for
(40)   Emit(HST[i]=)
(41)   foreach Service S in CloudServices[i]  begin
(42)       j = S.index
(43)       if(S is CLOUDHOSTING)
(44)          Emit(h_transfer[i][j] →H[i][j] | )
(45)       end if
(46)   end for
(47)   Emit(CLF[i]=)
(48)   foreach Service S in CloudServices[i]  begin
(49)     j = S.index
(50)     if(S is CLOUDFICATION)
(51)          Emit (f_transfer[i][j] →F[i][j] | )
(52)     end if
(53)   end
(54)   foreach Service S in CloudServices[i]  begin
(55)     j = S.index
(56)     if(S is CLOUDHOSTING)
(57)          Emit(H[i][j] = signal_client[i] → deployed[i] →
               SRECONNECT[i].)
(58)     elseif(S is CLOUDFICATION )
(59)          Emit(F[i][j] = signal_client[i] → deployed[i] →
               SRECONNECT[i].)
(60)     endif
(61)   end for
(62) end
```

۴-۲. تابع Synchronizer()

همان‌طور که در شبه کد تابع *Synchronizer()* تشریح شده است برای ایجاد هماهنگی میان فرآیندها، دو مجموعه فرآیندهای همگام‌سازی وجود دارد. مجموعه اول شامل فرآیندهای CONNECT[i][j] است. همان‌گونه که در بخش قبل بدان اشاره شد، برای هر مؤلفه مشتری C_j از COMP[i] تعریف می‌شود. این

فرآیند، فرآیند مهاجرت سرویس‌دهنده COMP[i] را با همه مشتریان آن هماهنگ می‌کند. به این ترتیب که COMP[i] یکی از اقدامات "h_transfer"یا"f_transfer" را برای مهاجرت به ابر و بعد از آن اقدام "signal_client[i]" را اجرا می‌کند. پس از آن COMP[j] به‌عنوان یک مؤلفه مشتری اقدام "start_reconnect[i][j]" و اقدامات بعد از آن را برای برقراری مجدد اتصال اجرا می‌کند (خط ۹ در شبه کد تابع *Synchronizer()* و خط ۱۷ در شبه کد تابع *AddComponent()*). مجموعه دوم از فرآیندهای همگام‌سازی ترتیب استقرار مؤلفه‌ها را کنترل می‌کند. برای انجام این هدف با توجه به هر فرآیند COMP[j] که باید منتظر استقرار فرآیند COMP[i] قبل از خود باشد، فرآیند WAIT[i][j] تولید می‌شود (خط ۱۱ تا ۱۴ در شبه کد تابع *Synchronizer()*).

```
(1)  Synchronizer (Components)
(2)  begin
(3)     foreach Component Comp in Components
(4)     begin
(5)       i = Comp.index
(6)       foreach ComponentClient in Comp.ClientComponents  begin
(7)          j = Client.index
(8)          Emit (CONNECT [i][j] = signal_client[i] →
(9)           start_reconnect [i][j] → CONNECT[i][j] )
(10)      end
(11)      foreach Component C in Comp.PreDeploy   begin
(12)         j = C.index
(13)         Emit(WAIT[j][i]=deployed[j]→wait[j][i]→WAIT[j][i])
(14)      end for
(15)    end for
(16) end
```

۴-۳. تابع AddService()

به‌ازای هر سرویس ابری، یک فرآیند برای ارائه سرویس ابری به مدل FSP اضافه می‌شود که در شبه کد تابع *AddService()* نشان داده شده است. همان‌طور که در بخش بعدی بدان اشاره می‌شود برای مهاجرت یک مؤلفه بر روی ابر باید اقداماتی از این مؤلفه با اقداماتی از سرویس‌دهنده ابری هم‌زمان گردد.

```
(1)  AddService (CloudServices)
(2)  begin
(3)     foreach Service S in CloudServices
(4)        Emit(Cloud_Service [S.index] = (service[S.index]
           →Cloud_Service [S.index].)
(5)  end
```

۴-۴. تابع Compose()

در آخرین بخش از الگوریتم تمام فرآیندهایی که قبلاً تشریح شد برای ایجاد یک طرح مهاجرت ترکیب و باهم همگام می‌شوند. در این تابع اجرای اقدام "service[S.index]" از هر فرآیند سرویس‌دهنده ابری با اجرای یکی از اقدامات "h_transfer[i][j]"یا "f_transfer[i][j]" از COMP[i] هم‌زمان شده است. (خطوط ۱۷ تا ۲۶ در شبه کد تابع *Compose()*)

```
(1)  Compose (Comp , CloudServices)
(2)  begin
(3)     foreach Component i in Comp
(4)        Emit (COMP[i] ||)
(5)     endfor
(6)     foreach Service s in CloudServices
(7)        Emit (S[s] || )
(8)     end for
(9)     foreachComponentClient in Comp.ClientComponents  begin
(10)       j = Client.index
(11)       Emit (CONNECT [i][j]||)
(12)    end for
(13)    foreachComponentC in Comp.PreDeploy  begin
(14)       j = C.index
(15)       Emit(WAIT[j][i] || )
(16)    end for
(17)    foreach Component C in Comp
(18)       foreach Service S inCloudServices[i]   begin
(19)          s= S.index
(20)          i= C.index
(21)          if (s is CLOUDHOSTING)
(22)             Emit(service[s]/h_transfer[i][s],)
(23)          elseif (s is CLOUDFICATION)
(24)             Emit(service[s]/f_transfer[i][s],)
(25)          end if
(26)       endfor
(27) end
```

۵. مثال عملی از پیاده‌سازی روش پیشنهادی

در این بخش از مثال ارائه شده در [۲۱] استفاده شده است. یک شرکت خدمات مالی تصمیم می‌گیرد برنامه‌های کاربردی محلی خود را به ابر مهاجرت دهد. برنامه‌ها از فناوری‌های مایکروسافت استفاده می‌کنند، اما برخی سامانه‌های قدیمی نیز مبتنی بر یونیکس هستند. برنامه کاربردی Expense اجازه می‌دهد تا کارکنان هزینه‌ها و پرداخت‌های شرکت را ثبت و پردازش کنند. خرابی‌های کوتاه‌مدت و چندباره برای کارکنان قابل‌تحمل است، اما عدم دسترسی طولانی مدت قابل‌قبول نیست. ثبت هزینه‌ها در آخرین روزهای قبل از پایان هر ماه توسط کارمندان، موجب بالا رفتن تقاضا می‌گردد. این در حالی است که زیرساخت نرم‌افزار تنها برای استفاده متوسط در نظر گرفته شده است. این نرم‌افزار به‌صورت محلی مستقر شده است و به دلیل اسکن تمام اسناد و مدارک، نیاز به ظرفیت ذخیره‌سازی با حجم بالا دارد. برنامه Expense مبتنی بر فناوری ASP.NET است و از احراز هویت ویندوز برای تأمین امنیت استفاده می‌کند. برای ذخیره تنظیمات کاربر نیز بر امکانات ASP.NET متکی است. برای پرس‌وجوی داده نیز از APIهای سرویس دایرکتوری استفاده شده است. این اطلاعات در سرویس‌دهنده SQL ذخیره می‌شود و رسیدها نیز در یک سامانه فایل ذخیره می‌گردند. برای حفظ خط‌مشی امنیتی

شرکت مؤلفه"User Profile" بر روی میزبان محلی باقی مانده و به ابر منتقل نمی‌گردد.

برنامه کاربردی مثال متشکل از ده مؤلفه همراه با هفت سرویس: H1، H2، H3، F2 ، F3، F4 و H4 به‌عنوان مثال مورد استفاده قرار گرفته است (H استفاده از روش میزبانی ابر و F استفاده از روش ابری‌سازی را نشان می‌دهد). شکل (۴) بیانگر معماری برنامه و شکل (۵) مدل برنامه های کاربردی را بر اساس مدل ویژگی نشان می دهد. همچنین وابستگی میان مؤلفه‌ها نیز در جدول (۲) ذکر شده است.

با استفاده از الگوریتم پیشنهادی مدلFSP برای مثال ما ایجاد شده است. به‌عنوان نمونه کد FSP ایجاد شده توسط تابع AddComponent() برای مؤلفه Active Directory به شکل زیر است:

COMP[6] = (wait[2][6]→wait[8][6]→**HST[6]**
| wait[2][6]→wait[8][6]→**CLF[6]**),

DEPLOYED[6] = (shutdown[6] → start_redeploy[6] → **COMP[6]**
| shutdown[6]→restart[6]→**DEPLOYED[6]**
| start_reconnect[2][6]→**CRECONNECT[2][6]**),

CRECONNECT[2][6] = (post_rebind[2][6]→**DEPLOYED[6]**
| post_adapt[2][6]→**DEPLOYED[6]**
| post_conv[2][6]→**DEPLOYED[6]**
|
post_compensate[2][6]→**DEPLOYED[6]**),

HST[6]=(h_transfer[6][6]→**H[6][6]**),

CLF[6]=(f_transfer[6][15]→**F[6][15]**),

H[6][6]=(signal_client[6]→deploy[6]→**SRECONNECT[2][6]**),

F[6][15]=(signal_client[6]→deploy[6]→**SRECONNECT[2][6]**),

SRECONNECT[2][6] =post_rebind[2][6]→**DEPLOYED[6]**
| post_adapt[2][6]→**DEPLOYED[6]**
| post_conv[2][6]→**DEPLOYED[6]**
|
post_compensate[2][6]→**DEPLOYED[6]**)

علاوه بر این، کد تولید شده با استفاده از تابع AddService() برای منابع ابری H1 و F2 به شکل زیر است:

Cloud_Service[3] = (service[3]→ **Cloud_Service**[3]).
Cloud_Service[4] = (service[4]→ **Cloud_Service**[4]).

در همین رابطه کدهای تولید شده برای هم‌زمانی میان مؤلفه‌ها با استفاده از تابع Synchronizer() به شکل زیر است:

CONNECT[1][2] = (signal_client[1]→start_reconnect[1][2]→ **CONNECT**[1][2]).

CONNECT [1][3] = (signal_client[1]→start_reconnect[1][3]→ **CONNECT** [1][3]).

WAIT[1][2] = (deployed[1] → wait[1][2] → **WAIT**[1][2]).

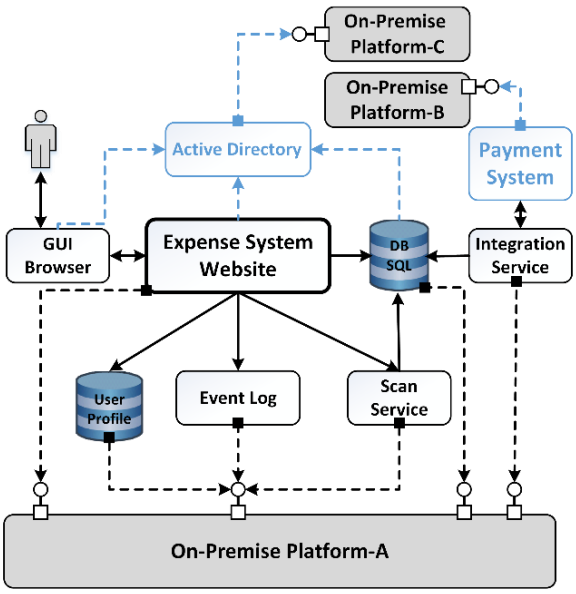
WAIT[1][3] = (deployed[1] → wait[1][3] → **WAIT**[1][3]).

و درنهایت، کد تولید شده با استفاده از تابع Compose() به شکل زیر است:

|| **Migration** = (COMP[1] || COMP[6] || COMP[4] || COMP[5] || COMP[7] || COMP[3] || COMP[2] || COMP[8] || COMP[9] || COMP[10] || CloudService[3] || CloudService[4] || CloudService[5] || CloudService[6] || CloudService[7] || CloudService[11] || CloudService[15] || CONNECT[1][3] || WAIT[2][1] || WAIT[4][1] || WAIT[5][1] || WAIT[6][1] || WAIT[7][1] || WAIT[8][1] || CONNECT[6][1] || CONNECT[6][4] || WAIT[2][6] || WAIT[8][6] || CONNECT[4][1] || WAIT[6][4] || WAIT[8][4] || CONNECT[5][1] || WAIT[8][5] || CONNECT[7][1] || WAIT[8][7] || WAIT[9][3] || CONNECT[2][1] || CONNECT[2][6] || WAIT[10][2])

/{service[3]/h_transfer[1][3],service[6]/h_transfer[1][6],service[6]/h_transfer[6][6],service[15]/f_transfer[6][15],service[3]/h_transfer[5][3],service[5]/f_transfer[5][5],service[6]/h_transfer[3][6],service[11]/h_transfer[3][11],service[3]/h_transfer[2][3],service[4]/f_transfer[2][4],service[3]/h_transfer[8][3],service[6]/h_transfer[9][6],service[11]/h_transfer[9][11],service[7]/h_transfer[10][7],}.

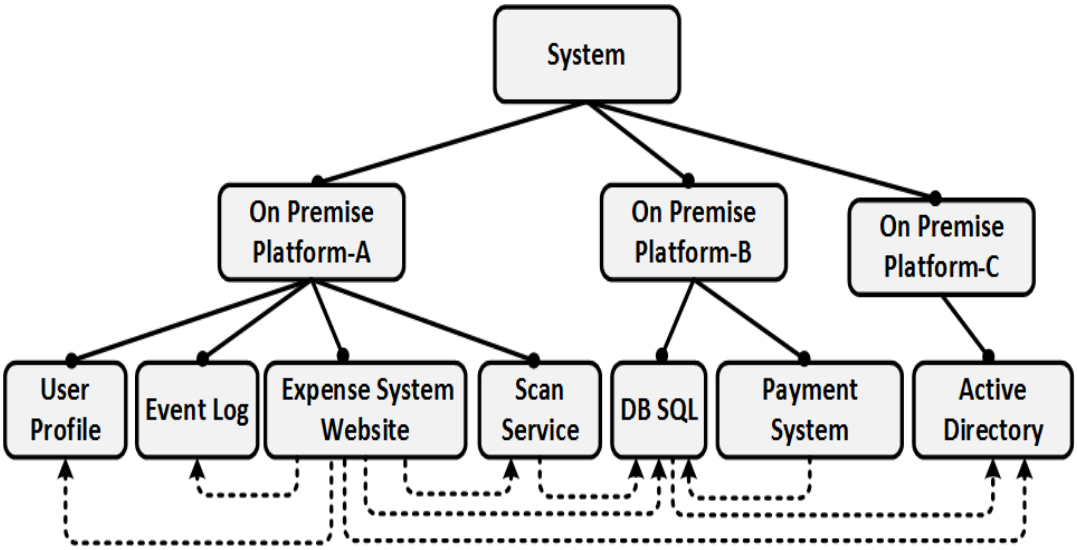
این مدل با استفاده از ابزار تحلیل سامانه انتقال برچسب‌گذاری شده اجرا و تحلیل‌شده و تعداد حالات و انتقال‌های FSP در جدول (۳) آمده است. می‌توان گفت از دیدگاه توسعه سامانه، در صورتی که سرویس‌های ابری و الگوهای مهاجرت جدید اضافه شوند، به همان ترتیب اقدامات جدید مستقر و به لیست اقدامات اضافه می‌شوند. بنابراین، این روش توسعه‌پذیر است. علاوه بر این، این مدل می‌تواند برای تحلیل و آزمایش امکان‌سنجی مسیر مهاجرت (با توجه به الزامات غیرعملیاتی و اهداف راهبردی سازمان) مورد استفاده قرار گرفته و هزینه هر مسیر قبل از استفاده و استقرار مؤلفه‌ها اندازه‌گیری شود. بنابراین این مدل می‌تواند یک مبنای رسمی از الگوریتم‌های پشتیبانی تصمیم برای کمک به مدیران برای انتخاب یک برنامه مهاجرت مناسب برای شرکت خود باشد.



شکل ۴. معماری برنامه‌های کاربردی مثال بخش۵ [۲۱]

جدول ۲. مؤلفه‌ها، منبع ابری و وابستگی‌های آن‌ها مربوط به شکل (۴)

Function → Component ↓	ID	PreDeploy	Server Components	Client Components	Cloud Resource	Cost
ESW	۱	۷،۸،۲،۴،۵،۶	۷،۲،۴،۵،۶	۳	H1,H2	۴۲/۴۰
Active Directory	۲	۱۰	-	۱/۶	H1,F2	۳۰/۱۵
Peyment System	۳	۹	۱	-	H2,H4	۳۵/۴۰
Scan Service	۴	۶/۸	۶	۱	-	-
Event Log	۵	۸	-	۱	F3,H1	۱۵/۱۵
DB SQL	۶	۲/۸	۲	۱/۴	F4,H2	۲۰/۲۰
User Profile	۷	۸	-	۱	illegal	۱۵
OP Platform-A	۸	-	-	-	H1	۵۰
OP Platform-B	۹	-	-	-	H2,H4	۶۰/۵۵
OP Platform-B	۱۰	-	-	-	H3	۳۵



شکل ۵. مدل برنامه‌های کاربردی مثال بخش۵

جدول ۳. مثالی از مدل ارائه شده در این مقاله

تعداد حالت	تعداد اقدامات	تعداد سرویس‌های ابری	تعداد مؤلفه‌ها
۹۳	۱۹۶	۷	۱۰

۶. پیاده‌سازی الگوریتم تولید خودکار کد FSP و مجری آن

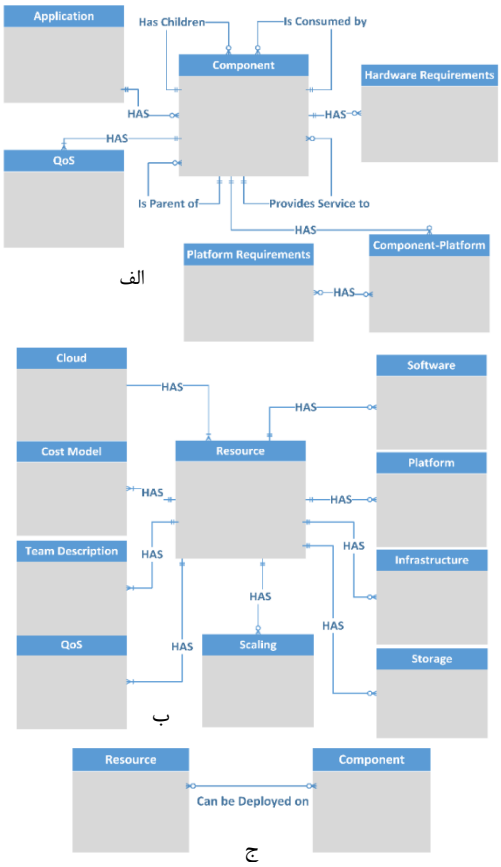
الگوریتم ارائه‌شده در بخش‌های قبلی برای تولید خودکار مدل FSP برای مهاجرت یک برنامه کاربردی به ابر در این بخش پیاده‌سازی شده است. همچنین اجراکننده کد FSP نیز طراحی و پیاده‌سازی شده است. برای پیاده‌سازی آن‌ها از زبان برنامه‌نویسیC#.Net2017 و پایگاه داده SQLServer2016 استفاده شده است. برای مدل‌سازی مفهومی نیز از نرم‌افزار MSVisio2016 بهره گرفته شده است.

۶-۱. طراحی مدل مفهومی پایگاه داده

همان‌گونه که در بخش‌های قبلی اشاره شد برای تولید یک طرح مهاجرت به مشخصات برنامه کاربردی و ارائه‌کنندگان خدمات ابری به‌عنوان ورودی مسئله نیازمند هستیم. مدل مفهومی این مشخصات طراحی و سپس در قالب جداول متعدد در پایگاه داده پیاده‌سازی شده‌اند. همان‌طورکه شکل (۶) نشان می‌دهد دو موجودیت اصلی به نام‌های مؤلفه و منبع رایانشی مشخص شده است که هرکدام دارای ویژگی‌هایی مختلفی می‌باشند. هر برنامه کاربردی از مؤلفه‌هایی تشکیل شده که مشخصات آن‌ها توسط موجودیت مؤلفه و ویژگی‌های آن نمایش داده می‌شود. همچنین موجودیت منبع رایانشی برای نمایش هرگونه منبع رایانشی اعم از نرم‌افزار، سخت‌افزار ، سکو ، زیرساخت ، انباره و حتی گروه پیاده‌سازی و استقرار استفاده می‌گردد. جدول (۴) حاوی توضیحی مختصر در رابطه با هر ویژگی است.

جدول ۴. موجودیت‌ها ، ویژگی‌های و شرح آن‌ها

موجودیت	ویژگی	توضیحات
مؤلفه	QoS	مشخصات کیفیت سرویس موردنیاز یک مؤلفه
	Component Platform	مشخصات سکوی یک مؤلفه
	Hardware Reuirement	مشخصات سخت‌افزار موردنیاز یک مؤلفه
	Componnet	لیست مشتریان یک مؤلفه
	Component	لیست سرویس‌دهندگان به یک مؤلفه
	Component	لیست مؤلفه‌های پیش‌نیاز یک مؤلفه
منبع رایانشی	Software	مشخصات نرم‌افزاری یک منبع رایانشی (SaaS)
	Platform	مشخصات سکوی یک منبع رایانشی (PaaS)
	Infrastrucure	مشخصات زیرساخت یک منبع رایانشی (IaaS)
	Storage	مشخصات انباره یک منبع رایانشی
	Scaling	مشخصات قابلیت مقیاس بندی منبع رایانشی
	Cost Model	هزینه استفاده از منبع رایانشی
	QoS	مشخصات کیفیت سرویس ، منبع رایانشی
	Team Describtion	مشخصات گروه استقرار منبع رایانشی

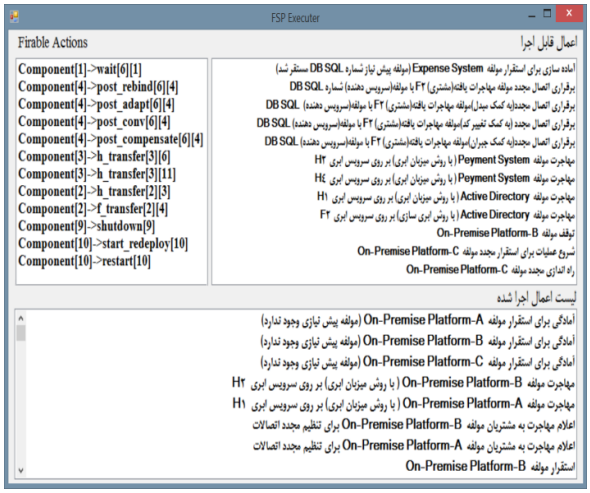


شکل ۶. مدل مفهومی الف) مؤلفه برنامه کاربردی ب) منبع ابری ج) ارتباط میان منبع ابری و مؤلفه برنامه کاربردی

۶-۲. طراحی و پیاده‌سازی پایگاه داده و اجراکننده FSP

پس از طراحی مدل مفهومی، پایگاه داده رابطه‌ای و اجراکننده مدل FSP در قالب یک سامانه جهت استفاده و ارزیابی عملی پیاده‌سازی شده‌اند. برای افزایش قابلیت حمل برنامه در ابتدا کد FSP طرح مهاجرت با توجه به مشخصات برنامه کاربردی و سرویس‌های ابری و استفاده از نرم‌افزار تولید کد FSP، ایجاد می‌گردد. این کد در دو قالب XML و Textبه‌عنوان کد میانی ذخیره می‌گردد. کد XML به‌عنوان ورودی اجرای کننده FSP عمل می‌کند و خروجی اجراکننده نیز مجموعه اقدامات قابل انجام در هر مرحله است. کد زیر بخشی از کد XML تولید شده برای FSP مؤلفه User Profile و شکل (۷) نیز نمایی از اجرای سامانه را نشان می‌دهد.

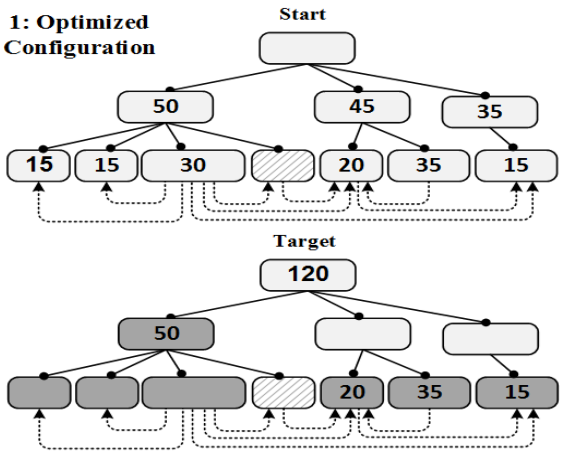
```
<StateMachine ID="7" Name="User Profile" Type="Component">
<Role>
<Left>COMP[7]</Left>
<Action>wait[8][7],signal_client[7],deploy[7]</Action>
<Right>DEPLOYED[7]</Right>
</Role>
<Role>
<Left>DEPLOYED[7]</Left>
<Action>shutdown[7],start_redeploy[7]</Action>
<Right>COMP[7]</Right>
</Role>
</StateMachine>
```



شکل ۷. نمایی از اجراکننده FSP

۷. نتایج و بحث

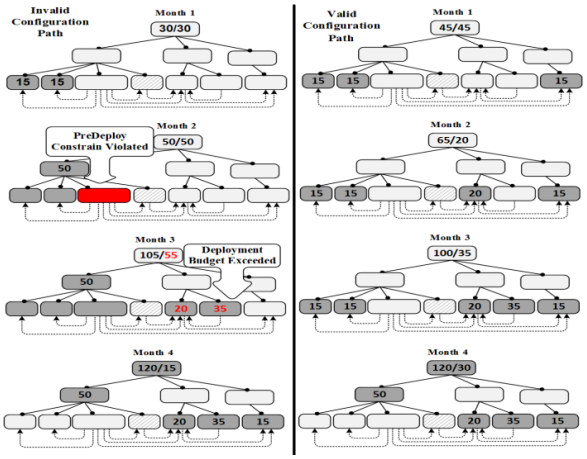
هر راه‌حل ممکن برای مهاجرت برنامه‌های کاربردی به ابر منجر به معرفی پیکربندی هدف مطابق با مجموعه‌ای از محدودیت‌ها شامل الزامات سازمان، برنامه کاربردی و ارائه‌دهندگان خدمات ابری است. برخی از آن‌ها از روش‌های بهینه‌سازی برای یافتن بهترین پیکربندی ممکن با توجه به محدودیت‌ها استفاده نموده‌اند. این روش‌ها در یک مرحله پیکربندی هدف را یافته و فرض می‌کنند امکان انتقال مستقیم از پیکربندی کنونی به آن وجود دارد. به‌عنوان نمونه در مثال شکل (۵) پس از مهاجرت هزینه بهینه استفاده از زیرساخت ابری به‌جای زیرساخت محلی ۱۲۵ هزار دلار است. شکل (۸) را ببینید.



شکل ۸. استفاده از روش بهینه‌سازی برای مهاجرت نرم‌افزار به ابر

در اغلب موارد قوانین و محدودیت‌ها مانع از انتقال مستقیم از حالت شروع به پیکربندی نهایی می‌شوند. به‌عنوان مثال فرض سازمان برای مهاجرت برنامه کاربردی به ابر به‌جای ۱۲۰ هزار دلار تنها یک بودجه دوره‌ای ۵۰ هزار دلاری برای توسعه سامانه دارد. بدین معنی که توسعه‌دهندگان نمی‌توانند در یک دوره تمام مؤلفه‌ها را به ابر منتقل کنند. با توجه به این محدودیت، توسعه‌دهندگان باید به‌صورت گام‌به‌گام مؤلفه‌ها را به ابر منتقل کنند. به‌عبارت‌دیگر در طول چندین دوره مجموعه‌ای از پیکربندی‌های میانی ایجاد می‌شود که برای دست‌یابی به پیکربندی هدف مورد استفاده قرار می‌گیرند. به‌عنوان مثال، آن‌ها می‌توانند تعدادی از مؤلفه‌ها را در دوره اول منتقل کنند و مؤلفه‌های باقی مانده را در دوره‌های دوم و سوم منتقل کنند و بقیه مؤلفه‌های مورد نظر تا رسیدن به پیکربندی نهایی، در دوره چهارم منتقل نمایند. این دنباله از فعالیت‌ها یک مسئله مهاجرت گام‌به‌گام نامیده می‌شود.

یک مسئله اساسی این است که توسعه‌دهندگان نمی‌دانند با توجه به محدودیت‌ها، در هر مرحله کدام مؤلفه را برای انتقال به ابر انتخاب کنند. به‌عنوان مثال همان‌طور که در شکل (۹) نشان داده شده است ، در صورتی که توسعه‌دهندگان در ماه دوم برای گسترش پیکربندی مؤلفه ESW را بدون در نظر گرفتن مؤلفه پیش‌نیاز آن مانند DB SQL , Active Directory برای انتقال به ابر انتخاب کنند قانون پیش استقرار (PreDeploy) را نقض نموده‌اند. به‌طور مشابه در ماه سوم انتخاب هم‌زمان مؤلفه‌های Payment System و DB SQL نیازمند بودجه ۵۵ هزار دلاری است که از حداکثر بودجه ماهیانه تجاوز می‌کند. این پیکربندی نیز یکی از قوانین انتقال را نقض می‌کند و بر همین اساس پیکربندی معتبری نیست. بنابراین توسعه‌دهندگان نه‌تنها باید به محدودیت‌های خود در تغییراتی که می‌توانند در یک دوره انجام دهند، توجه کنند بلکه باید اطمینان حاصل کنند اِعمال تغییرات پیکربندی معتبری را در پایان هر دوره ایجاد می‌کند. همچنین توسعه‌دهندگان نمی‌توانند پیکربندی‌های میانی نامعتبر را به‌منظور گذار از طریق آن‌ها برای رسیدن به پیکربندی نهایی انتخاب کنند.



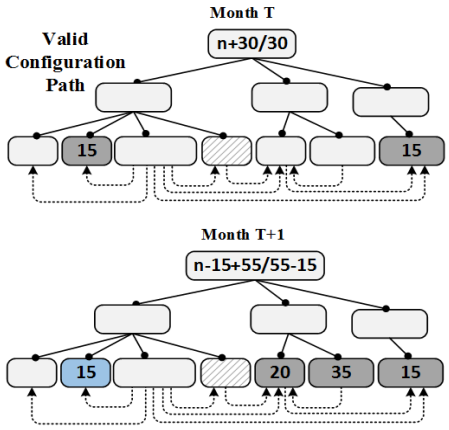
شکل ۹. مسیرهای نامعتبر و معتبر مهاجرت -گره ریشه "هزینه هر مرحله / هزینه صرف شده با احتساب این مرحله" مربوط به هر گام را نشان می‌دهد

در مطالعات صورت‌گرفته در رابطه با مسائل مهاجرت چندمرحله‌ای معمولاً مسیری که منجر به انتقال نرم‌افزار به ابر می‌شود از ابتدا مشخص است زیرا این مسائل فرض می‌کنند تمامی عوامل مؤثر در تصمیم‌گیری‌ها به‌صورت ایستا و از ابتدا مسئله تعیین می‌شوند، اما در عمل این‌گونه نیست. محیط پویای سازمان‌ها بر مقادیر شاخص‌های مؤثر بر تصمیم‌گیری تأثیر می‌گذارند و مقادیر آن‌ها را طول زمان تغییر می‌دهند. تغییر اولویت‌ها و محدودیت‌ها در انتقال به ابر از آثار این تغییر سیاست است. به‌عنوان مثال در حالت معمول و بر اساس برنامه‌ریزی‌های قبلی مقرر بوده شرکت X خدمات A و B را در سه ماه آینده به ابر منتقل نماید. حال بر اساس شرایط اقتصادی ، سیاسی و ...، شرکت X باید حداکثر تا دو ماه آینده سرویس‌های خود را بر روی ابر منتقل کند. درنتیجه نقشه‌ای که از قبل برای انتقال سرویس‌ها به ابر طراحی‌شده باید اصلاح گردد تا اهداف سازمان را تأمین نماید که این همان رویکرد ارائه شده در این مقاله است.

پیچیدگی دیگر مسائل پیکربندی چندمرحله‌ای این است که توسعه‌دهندگان ممکن است نیازمند پیش‌بینی و موازنه بین انتخاب‌های مختلف در طول مسیر مهاجرت شوند. به‌عنوان مثال همان‌طور که در شکل (۹) نشان می‌دهد در ماهی که هزینه توسعه ۵۰ هزار دلار است امکان انتقال مؤلفه Payment System به همراه مؤلفه‌های پیش‌نیاز وجود ندارد و بر همین اساس این مؤلفه در ماه سوم بر روی ابر منتقل شده است. حال اگر پس از گذشت یک ماه سیاست سازمان بر انتقال هرچه زودتر مؤلفه

Payment System بر روی ابر باشد بنابراین، مسیر انتقال تعیین شده در ماه دوم باید تغییر کند. یک راه‌حل موجود که در شکل (۱۰) نشان داده شده ، این است که ابتدا مؤلفه Event log از روی ابر به سرورهای محلی منتقل و به‌تبع آن منابع سازمان بر روی ابر آزادشده و پس از افزایش به مؤلفه Payment System اختصاص یابد. درنهایت میزان هزینه استفاده از خدمات ابری در ماه دوم برابر ۴۰ هزار دلار (۲۰+۳۵-۱۵) است.

با توجه به مطالب فوق مزایا و معایب رویکردهای فوق در جدول (۵) به‌طور خلاصه آورده شده است. رویکرد گام‌به‌گام وفقی همان رویکرد ارائه‌شده در این مقاله است که مهم‌ترین مزیت آن پشتیبانی از خودکارسازی طرح مهاجرت در محیط‌های سازمانی پویا است.



شکل ۱۰. تغییر مسیر پیکربندی انتقال نرم‌افزار به ابر با توجه به اولویت‌های سازمان

جدول ۵. مقایسه میان رویکردهای قبلی و رویکرد ارائه شده در این مقاله

رویکرد	روش	نمونه کارهای تحقیقاتی	مزایا	معایب
استفاده از افراد خبره برای ایجاد طرح مهاجرت	طراحی طرح مهاجرت به ابر با استفاده از تجربه خبرگان و به‌صورت دستی	[۲]Khajeh-Hosseini,A.,et al. [۳] Andrikopoulos,V.,et al. [۵] Maenhaut, P. J. , et al. [۱۷] Tran, V. , et al. [۱۸] Cunha, M. , et al.	• اطمینان از عملی بودن برنامه مهاجرت برای برنامه کاربردی با پیچیدگی پائین • سادگی اجرای الگوریتم	• عدم ارائه راه‌حل بهینه به دلیل وجود تعداد نمایی پیکربندی‌ها و مسیرهای میانی بالقوه • عدم پشتیبانی از طرح مهاجرت در محیط‌های سازمانی پویا
تک‌مرحله‌ای و ایستا: استفاده از یکی از روش‌های بهینه‌سازی برای جستجوی یک پیکربندی بهینه مطابق با الزامات هدف	ایجاد یک پیکربندی مطابق با یک مجموعه خاص از محدودیت‌ها. این روش‌ها در یک مرحله پیکربندی نهایی را یافته و فرض می‌کنند امکان انتقال مستقیم از پیکربندی اولیه به پیکربندی نهایی وجود دارد.	[۶] Saripalli, P. , et al. [۷] Menzel, M. , et al. [۸] Frey, S. , et al. [۹] Sahandi, R. , et al. [۱۰] Dieter, F. , et al. [۱۱] Li, A. , et al. [۱۲] Omerovic, A. , et al. [۲۲] Leymann, F. , et al. [۲۳] Megahed, A. , et al. [۲۴] García-Galán , J. , et al.	• یافتن پیکربندی بهینه در سریع‌ترین زمان • مناسب برای برنامه‌های کاربردی ساده	• عدم اطمینان از عملی بودن برنامه مهاجرت • عدم پشتیبانی از مهاجرت گام‌به‌گام • عدم پشتیبانی از طرحی برنامه مهاجرت خودکار در چندین مرحله • قابلیت تغییر طرح بر اساس شرایط پویای سازمان را ندارد
گام‌به‌گام : توسعه‌دهندگان در طول چندین مرحله مجموعه‌ای از پیکربندی‌های میانی را ایجاد می‌کنند که برای دستیابی به پیکربندی هدف مورد استفاده قرار می‌گیرند.	یک مسئله مهاجرت به ابر ممکن است بدون انتخاب هیچ مؤلفه‌ای آغاز و در گام‌های میانی با انتخاب یک یا چند مؤلفه برای مهاجرت ادامه یابد تا درنهایت به یک پیکربندی نهایی هدف برسد.	[۱۹] Mendonca, N. C., et al. [۲۰] Fehling, C. , et al. [۲۱] Jamshidi, P. , et al.	• اطمینان از عملی بودن برنامه مهاجرت • پشتیبانی از مهاجرت گام‌به‌گام. • پشتیبانی از روش‌های مهاجرت خودکار به ابر • مناسب برای برنامه‌های کاربردی نسبتاً پیچیده	• قابلیت تغییر طرح بر اساس شرایط پویای سازمان را ندارد • پیچیدگی الگوریتم نسبت به روش تک‌مرحله‌ای و ایستا
گام‌به‌گام وفقی: توسعه‌دهندگان انتخاب‌های مختلف در طول مسیر مهاجرت را با شرایط پویای سازمان وفق می‌دهند	مسیر مهاجرت تعیین شده بر اساس تغییر اهداف سازمان تغییر می‌کند	-	• پشتیبانی از خودکار سازی طرح مهاجرت در محیط‌های سازمانی پویا • اطمینان از عملی بودن برنامه مهاجرت • پشتیبانی از مهاجرت گام‌به‌گام. • پشتیبانی از روش‌های مهاجرت خودکار به ابر • مناسب برای برنامه‌های کاربردی پیچیده	• پیچیدگی زیاد الگوریتم نسبت به دو رویکرد قبلی

۸. نتیجه‌گیری

در این مقاله، یک مدل‌سازی رسمی از مسئله مهاجرت ابر به‌عنوان یک فرآیند FSP ترکیبی ارائه شده است. این مدل‌سازی رسمی می‌تواند به‌عنوان پایه‌ای برای توسعه سامانه‌های پشتیبانی تصمیم درزمینه مهاجرت ابر استفاده شود. به‌نحوی‌که پس از ایجاد FSP طرح مهاجرت، اجرای آن به‌صورت خودکار پیش خواهد رفت و کمترین نیاز برای انجام امور به‌صورت دستی را خواهد داشت. مدل ترکیبی FSP تولیدشده توسط یک نرم‌افزار طراحی‌شده به‌طوری اجرا می‌شود که در هر گام مجموعه اقدامات قابل‌اجرا توسط گروه استقرار پیشنهاد گردد. بدین ترتیب سامانه طراحی‌شده از طریق یک برنامه مهاجرت گام‌به‌گام از طرح ایجاد شده برای مهاجرت به ابر پشتیبانی می‌کند.

۹. مرجع‌ها

[1] Rahimdel Meybodi, M. ; Amiri , A.; Karbasian, M. “Evolutionary Stable Strategies to Defend of Sensitive Systems with False Attacks and Reliability Approach”; Advanced Defence Sci. Technol. 2017, 8, 339–348. (In Persian)

[2] Khajeh-Hosseini, A.; Greenwood, D.; Sommerville, I. “Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS”; Proc. IEEE 3rd Int. Conf. Cloud Comput. 2010, 450–457.

[3] Andrikopoulos, V.; Binz, T.; Leymann, F.; Strauch, S. “How to Adapt Applications for the Cloud Environment”; Comput. 2013, 95, 493–535.

[4] Buyya, R.; Yeo, C. S.; Venugopal, S.; Broberg, J.; Brandic, I. “Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility”; Futur. Gener. Comput. Syst. 2009, 25, 599–616.

[5] Maenhaut, P. J.; Moens, H.; Ongenae, V.; De Turck, F. “Migrating Legacy Software to the Cloud: Approach and Verification by Means of Two Medical Software Use Cases”; Softw. Pract. Exp. 2016, 46, 31–54.

[6] Saripalli, P.; Pingali, G. “MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds”; Proc. IEEE 4th Int. Conf. Cloud Comput. CLOUD 2011, 316–323.

[7] Menzel, M.; Ranjan, R. “CloudGenius: Decision Support for Web Server Cloud Migration”; Proc. 21st Int. Conf. World Wide Web - WWW’12 2012, 979–988.

[8] Frey, S.; Hasselbring, W. “The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications”; Int. J. Adv. Softw. 2011, 4, 342–353.

[9] Sahandi, R.; Alkhalil, A.; Opara-Martins, J. “Cloud Computing from SMEs Perspective : A Survey-Based Investigation”; J. Inf. Technol. Manag. 2013, 24, 1–12.

[10] Dieter, F.; Shoeib, A. “Key Decision-Making Phases and Tasks for Outsourcing Information Technology,” Pac. Asian. Conf. Inf. Syst. PACIS 2000, 774–784.

[11] Li, A.; Yang, X.; Kandula, S.; Zhang, M. “CloudCmp: Comparing Public Cloud Providers”; IMC’10 Proc. 10th ACM SIGCOMM Conf. Internet Meas. 2010, 1–10.

[12] Omerovic, A.; Munt, V.; Matthews, P.; Gunka, A. “Towards a Method for Decision Support in Multi-cloud Environments”; Fourth Int. Conf. Cloud Comput. 2013, 162–168.

[13] Fekete, A. “FSP Lectures”; http://www.it.usyd.edu.au/~fekete/ info2810/INFO2810fsp.pdf, 2004.

[14] Liu, T.; Lu, T.; Wang, W.; Wang, Q.; Liu, Z.; Gu, N.; Ding, X. “SDMS-O: A Service Deployment Management System for Optimization in Clouds While Guaranteeing Users’ QoS Requirements”; Futur. Gener. Comput. Syst. 2012, 28, 1100–1109.

[15] Chauhan, M. A.; Babar, M. A. “Migrating Service-Oriented System to Cloud Computing: an Experience Report”; Proc. IEEE 4th Int. Conf. Cloud Comput. CLOUD 2011, 404–411.

[16] Church, P.; Mueller, H.; Ryan, C.; Gogouvitis, S. V.; Goscinski, A.; Tari, Z. “Migration of a SCADA System to IaaS Clouds – a Case Study”; J. Cloud Comput. 2017, 6, 1–12.

[17] Tran, V.; Keung, J.; Liu, A.; Fekete, A. “Application Migration to Cloud: a Taxonomy of Critical Factors”; Proc. Int. Work. Softw. Eng. Cloud Comput. 2011, 22–28.

[18] Cunha, M.; Mendoņa, N.; Sampaio, A. “Investigating the Impact of Deployment Configuration and User Demand on a Social Network Application in the Amazon EC2 Cloud”; Proc. 3rd IEEE Int. Conf. Cloud Comput. Technol. Sci. 2011, 746–751.

[19] Mendonca, N. C. “Architectural Options for Cloud Migration”; IEEE Comput. Long. Beach. Calif. 2014, 47, 62–66.

[20] Fehling, C.; Leymann, F.; Ruehl, S. T.; Rudek, M.; Verclas, S. “Service Migration Patterns: Decision Support and Best Practices for the Migration of Existing Service-Based Applications to Cloud Environments”; Proc. IEEE 6th Int. Conf. Serv. Comput. Appl. SOCA 2013, 9–16.

[21] Jamshidi, P.; Pahl, C.; Mendonça, N. C. “Pattern-Based Multi-Cloud Architecture Migration”; Softw. Pract. Exp. 2017, 47, 1159–1184.

[22] Leymann, F.; Fehling, C.; Mietzner, R.; Nowwak, A.; Dustdar, S. “Moving Applications to the Cloud: An Approach Based on Application Model Enrichment”; Int. J. Coop. Inf. Syst. 2011, 20, 307–356.

[23] Megahed, A.; Nazeem, A.; Yin, P.; Tata, S.; Nezhad, H. R. M.; Nakamura, T. “Optimizing Cloud Solutioning Design”; Futur. Gener. Comput. Syst. 2019, 86–95.

[24] García-Galán, J.; Trinidad, P.; Rana, O. F.; Ruiz-Cortés, A. “Automated Configuration Support for Infrastructure Migration to the Cloud”; Futur. Gener. Comput. Syst. 2016, 55, 200–212.

[25] Alkhalil, A.; Sahandi, R.; John, D. “Migration to Cloud Computing: a Decision Process Model”; Cent. Eur. Conf. Inf. Intell. Syst. 2014, 154–163.

[26] Bushehrian, O.; Ghaedi, H.; Ghanbari Baghnavi, R. “Automated Transformation of Distributed Software Architectural Models to Finite State Process”; Int. J. Comput. Sci. Eng. 2010, 2, 3120–3125.