الگوریتمی برای یافتن ترکیبی بهینه از سرویس‌ها

محمدکاظم سپهری‌فر، دکترکامران زمانی‌فر، فرهاد مردوخی

گروه کامپیوتر دانشکده مهندسی دانشگاه اصفهان

{mksepehrifar, zamanifar, fmardukhi}@eng.ui.ac.ir

چکیده - امروزه سیستم‌های سرویس‌گرا به‌خاطر امکان فعالیت در محیط‌های توزیعی ناهمگن از اهمیت ویژه‌ای برخوردارند. کاربر این‌گونه سیستم‌ها از سرویس‌هایی که مولفه‌های سیستم ارائه می‌دهند، استفاده می‌کند. نیازهای کاربر در اغلب اوقات، توسط سرویس‌های منفرد موجود رفع نمی‌گردد و می‌بایست با ترکیب تعدادی از سرویس‌ها به سرویس مورد نظر دست یافت.

در این نوشتار روشی پویا برای یافتن بهترین ترکیب بر اساس کیفیت سرویس و با توجه به هم‌خوانی سرویس‌های شرکت‌کننده، ارائه می‌گردد. این روش گوناگونی واسط سرویس‌ها را در نظر گرفته و عملی بودن ترکیب را در هر نقطه از اجرا تشخیص می‌دهد.

بدین ترتیب روشی با سرعت بالا و مصرف کم حافظه معرفی می‌گردد که قادر به یافتن پویای ترکیب بهینه‌ی سرویس‌ها با در نظر گرفتن هم‌خوانی واقعی واسط‌هاست.

کلید واژه- بهینه‌سازی، سرویس مرکب، کیفیت سرویس

۱- مقدمه

مسائل دنیای واقعی دارای پیچیدگی زیادی هستند پس برای حل آنها توسط کامپیوتر نیازمند سیستمی با دامنه‌ی اطلاعاتی وسیع و قدرت پردازشی بالا هستیم که این ویژگی‌ها را در سیستم‌های توزیع‌شده می‌توانیم بیابیم. تابه‌حال معماری‌های مختلفی برای سیستم‌های توزیع‌شده پیشنهاد شده است که سیستم‌های سرویس‌گرا[1] به‌خاطر فعالیت در محیط‌های ناهمگن[2] از جمله کاربردی‌ترین آن‌ها هستند.[۱; ۲] وب و گرید معنایی دو گونه از سیستم‌های سرویس‌گرا هستند که به خاطر حضور در اینترنت از عمومیت بالایی بهره‌مندند و به همین سبب، به اطلاعات زیادی دسترسی دارند. [۳]

یکی از مسائل مطرح در زمینه‌ی این سیستم‌ها چگونگی ترکیب سرویس‌ها و نحوه‌ی مدیریت آن‌هاست. تاکنون روش‌های مختلفی برای عمل ترکیب ارائه شده است [۴; ۵; ۶; ۷] که در این نوشتار به‌دنبال راهی برای مدیریت ترکیب سرویس‌ها با توجه به لایه‌های معماری سرویس‌گرا هستیم [۲].

در ادامه، ابتدا در بخش ۲- نگاهی به سیستم‌های سرویس‌گرا و تعاریف مرتبط با آن می‌پردازیم و از تعاریف این قسمت در بخش‌های بعدی استفاده می‌کنیم. در بخش ۳-، راهی برای ترکیب بهینه‌ی سرویس‌ها به‌صورت پویا معرفی کرده و پس از ارزیابی اولیه‌ی آن، در بخش ۴- به طرز تجربی با نتایج آن آشنا می‌شویم.

۲- ترکیب سرویس‌ها

واحد سازنده[3]ی سیستم‌های سرویس‌گرا، سرویس می‌باشد. سرویس‌ها را می‌توان ترکیب کرده و سرویسی با ارزش افزوده[4] تولید کرد.

[1] service oriented systems
[2] heterogeneous
[3] building block

۲-۱- سرویس

همانگونه که از نام سرویس‌ها بر می‌آید به منظور رساندن خدمتی به خدمت‌گیرنده استفاده می‌شوند (خدمت‌گیرنده کاربر انسانی یا نرم‌افزاری است که از خدمت‌رسان[5] بهره می‌برد). خدمت‌گیرنده توسط اطلاعات فراداده‌ای[6] که به هر سرویس منتسب است، سرویس مورد نظر خود را می‌یابد. البته با افزایش سرویس‌ها و بزرگی فضای جستجو بهتر است از واسطه[7]هایی برای ثبت سرویس‌ها استفاده شود تا خدمت‌گیرندگان با سرعت بیش‌تر به سرویس با مشخصه‌های مناسب‌تر دست یابند.

هر سرویس دارای سه مشخصه‌ی کیفیت، واسط[8] و کارکرد[9] است که در ادامه به‌ترتیب به آن‌ها خواهیم پرداخت.

کیفیت هر سرویس بسته به نحوه‌ی انجام نیازهای وظیفه‌مندی و غیروظیفه‌مندی[10] در نظر گرفته می‌شود. مثلا نیاز وظیفه‌مندی در یک سرویس می‌تواند پیدا کردن خروجی مناسب باشد درحالی‌که هرچه سریع‌تر پیدا کردن آن، جزء نیازهای غیروظیفه‌مندی است. کاربر با توجه به معیارهای خود، به برآورده شدن هر نیازی امتیازی می‌دهد. مثلا کاربری حاضر است در قبال یافتن خروجی مناسب‌تر بیشتر منتظر بماند.

برای مقایسه‌ی کیفی سرویس‌ها، QoS کیفیت را به صورت کمی در می‌آورد. رفع هرنیازی برای کاربر امتیازی دارد که با آن کیفیت کلی سرویس ارزیابی و در مقایسه‌ها استفاده می‌شود. نحوه‌ی محاسبه‌ی QoS به‌صورت فرمول ۱ است.[۸]

(۱)
$$QoS(s) = \sum w_i * Score_i(s)$$

در این فرمول، کاربر به برآورده شدن نیاز iام امتیاز w_i داده و سرویس s با کیفیت $Score_i$ توانسته است این نیاز را مرتفع نماید.

مشخصه‌ی دیگر سرویس‌ها داشتن واسط[11] است. هر سرویس دارای دو واسط می‌باشد که واسط ورودی پارامترهای ورودی و واسط خروجی پارامترهای خروجی سرویس را مشخص می‌کند. دو سرویس را هم‌خوان[12] می‌نامیم اگر پارامترهای خروجی یکی بر پارامترهای ورودی دیگری منطبق باشد. یک انجمن[13]، مجموعه‌ای از سرویس‌هایی است که با هم‌خوان هستند.

به‌خاطر استفاده‌ی جامع از سرویس‌ها، اغلب آن‌ها به گونه‌ای طراحی می‌شوند که مشخصه‌ی کارکرد آن‌ها بتواند مسائل ساده را برای انجام عملیات اتمیک حل کند. برای حل مسائل پیچیده‌تر، توسعه دهندگان نرم‌افزار[14] ترکیبی مناسب از آن‌ها را برگزیده و با اجرای یک‌به‌یک آن‌ها در توالی خاصی به حل مسئله‌ی طرح‌شده می‌رسند. این توالی در برنامه‌ی اجرای مرتبط با مسئله مشخص می‌شود.

۲-۲- برنامه‌ی اجرا[15]

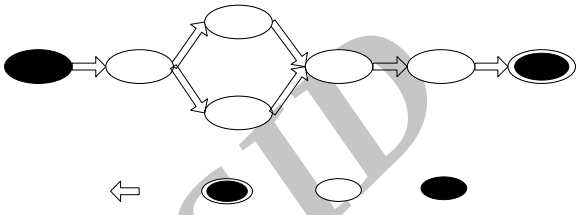
عملیاتی که برای حل یک مسئله انجام می‌پذیرد، عمل مرکب نام دارد. برای انجام هر عمل مرکبی عمل‌ها[16] با ترتیب ویژه‌ای اجرا می‌گردند. در برنامه‌ی اجرا عمل‌های تشکیل‌دهنده‌ی عمل مرکب به همراه ترتیب قرارگیری نمایش داده می‌شود.

مثلا در مورد سفر[17]، به‌دنبال برنامه ریزی صحیح و بهینه برای مسافرت هستیم. در این مسئله می‌بایست هماهنگی‌هایی بین زمانبندی‌های مختلف برای ساخت مجموعه‌ای از انتخاب‌های به‌هم‌وابسته صورت پذیرد. نیاز دسترسی به اطلاعات فراگیر و مرتبط نیز ما را به سمت استفاده از سیستم سرویس‌گرا برای حل این مسئله رهنمون می‌سازد. [۹]

[4] value added
[5] server
[6] meta data
[7] service broker
[8] interface
[9] functionality
[10] functional & non functional requirement
[11] interface
[12] match
[13] community
[14] software developers
[15] execution plan
[16] tasks
[17] travel

برنامه‌ی اجرا به‌گونه‌ای طراحی می‌شود که مراحل رفتن از مبدا به مقصد مسافر، اقامت و گردش در مقصد و مراحل بازگشت به مبدا او را مشخص نماید. برنامه‌ی اجرا در مساله‌ی مسافرت همان برنامه‌ی سفر[18] است که با بررسی شرایط مسافر و محیط اجرا برای کاربر تهیه می‌شود.

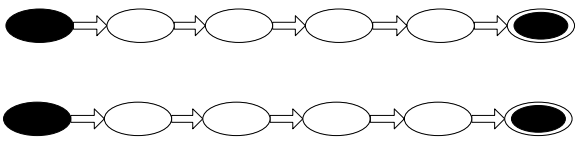
بخشی از یک برنامه‌ی سفر در شکل ۱ نشان داده شده است.



شکل ۱: یک برنامه‌ی سفر فرضی

در این برنامه، اعمال و توالی انجام آن‌ها قابل مشاهده است. پس از عمل t1 یکی از دو عمل t2 و t3 انجام می‌شود، به این‌گونه برنامه‌ها که حالت‌هایی از «یا»ی منطقی در آن‌هاست، برنامه‌های اجرای غیر خطی گوییم. این در حالیست که در برنامه‌ی خطی پس از هر عمل فقط یک عمل انجام می‌پذیرد.

به سادگی برنامه‌های غیر خطی را با شکستن آن‌ها به خطی تبدیل می‌کنیم.[۱۰; ۱۱] مثلا برنامه‌ی شکل ۱ را می‌توان به دو برنامه‌ی شکل ۲ تبدیل کرد. بنابراین می‌توان هر برنامه‌ای را خطی در نظر گرفت و عملیات مختلف بعدی را با فرض خطی بودن برنامه به انجام رساند.



شکل ۲: تبدیل برنامه‌ی اجرای غیر خطی به خطی

هنگامی که به‌دنبال اجرای یک سرویس ترکیبی برای انجام یک عمل مرکب هستیم، ابتدا برنامه‌ی اجرای آن را طرح کرده[19] سپس برای تک‌تک عمل‌ها، سرویس مورد نظر را می‌یابیم. برای هر عمل در فضای سیستم ممکن است چندین سرویس وجود داشته باشد که آن‌ها را سرویس نامزد[20] می‌نامیم. اجرای طبق‌برنامه‌ی سرویس‌ها باعث اجرای سرویس مرکب می‌شود.

۲-۳- سرویس مرکب[21]

در فضای سیستم، برای هر عمل، سرویس‌های نامزد زیادی توسط تامین‌کنندگان[22] مختلف ارائه می‌شود. که انتخاب سرویس مناسب از بین آن‌ها توسط کاربر کاری دشوار و تقریبا ناممکن است[۹; ۴]، پس با ترکیب خودکار سرویس‌ها[23]، یک سیستم کامپیوتری می‌تواند از انبوه سرویس‌ها برای هر عمل سرویسی مناسب را در کم‌ترین زمان و بیشترین دقت برگزینند.

برای هر عمل در برنامه‌ی اجرا سرویسی را برمی‌گزینیم که ویژگی کارکرد آن حاکی از توانایی‌اش به انجام آن عمل است و نیز از واسط متناسب با سرویس‌های مجاور بهره‌مند باشد زیرا وسعت سیستم‌های سرویس‌گرا و وجود تامین‌کنندگان متعدد باعث ایجاد سرویس‌های همکار با واسط‌های مختلف می‌گردد و به هم‌خوان بودن آن‌ها هنگام ترکیب باید توجه گردد.

در هنگام ساخت سرویس از میان حالتهای مختلف، حالتی انتخاب می‌گردد که علاوه بر انطباق دو ویژگی کارکرد و واسط، با توجه به معیارهای کاربر، سرویس حاصله دارای بهترین کیفیت (بیشترین QoS) نیز باشد تا در نهایت یک سرویس مرکب بهینه[24] داشته باشیم.

بهینه بودن را به دو صورت محلی و سراسری[25] می‌توان در نظر گرفت. در بهینگی محلی برای هر عمل سرویسی با بالاترین QoS را از مجموع سرویس‌های همکار برگزینیم. این روش حریصانه[26] عمل کرده پس

[18] itinerary

[19] تا کنون روش‌هایی برای تولید برنامه‌ی اجرا ذکر شده است [۱۵; ۱۶]
[20] candidate service
[21] composed service
[22] providers
[23] automatic service composition
[24] optimized
[25] local & global optimization
[26] greedy

سرعت بالایی دارد ولی ضمانتی بر بهینه بودن کل سرویس نمی‌کند. در بهینگی سراسری دنبال سرویس‌هایی برای هر عمل می‌گردیم که از کنار هم قرار دادن آن‌ها نهایتا QoS سرویس مرکب، بالاترین مقدار ممکن را داشته باشد.

لازم به ذکر است که QoS کل سرویس مرکب برابر مجموع QoSهای سرویس‌های تشکیل‌دهنده‌ی سرویس مرکب است.

انتخاب بهترین حالت می‌تواند در زمان طراحی انجام شده و کلیه‌ی سرویس‌های مورد نیاز یک‌جا جمع‌آوری شود (ایستا[27]) یا این‌که در زمان اجرا صورت پذیرد که در این حالت سرویس‌های متناسب هر عمل را یافته سپس سراغ یافتن سرویس برای عمل بعدی می‌رویم (پویا[28]). به‌خاطر وجود تغییر در کیفیت و تعداد سرویس‌ها، انتخاب‌های پویا واقعی‌تر بوده و نتایج عملی‌تری به‌بار می‌آورند. [۱۲]

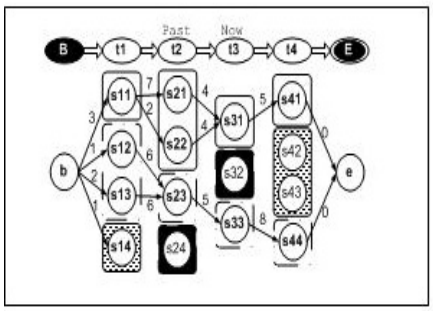
در ادامه با روشی برای یافتن پویای ترکیب بهینه به‌منظور ساخت سرویس مرکب آشنا می‌شویم.

۳- یافتن ترکیب بهینه

حال می‌بایست با توجه به برنامه‌ی اجرای طراحی شده، سرویس‌های مربوط به هر عمل را در زمان اجرا بیابیم. در این‌جا روشی ارائه می‌گردد که در آن سرویس‌های موجود در فضای سیستم را به عنوان گره‌های گراف در نظر گرفته و متناظر با ارتباط عمل‌ها، یال‌های این گراف را شکل می‌دهیم. پس از ساخت گراف، با جستجویی بین مسیرهای موجود در آن، بهترین مسیر را یافته و مجموعه سرویس‌های واقع در آن مسیر را به عنوان بهترین ترکیب معرفی می‌کنیم. در ادامه به چگونگی ساخت گراف و جستجو در آن می‌پردازیم.

۳-۱- ساخت گراف

برای هر عمل در برنامه‌ی اجرا، کلیه‌ی سرویس‌های موجود از واسطه‌ی سیستم تقاضا می‌شود. واسطه نیز آن‌ها را در قالب انجمن‌های مختلف به درخواست‌کننده ارائه می‌دهد. در شکل ۳ یک برنامه‌ی اجرای فرضی نشان داده شده است. در این شکل عمل‌های برنامه به‌صورت بیضی در بالا و سرویس‌های نامزد متناظر با هر عمل، به‌صورت ستونی زیر آن عمل، مشخص شده‌اند. سرویس‌های هم‌انجمن نیز در مستطیل‌های یک‌شکل محصور گشته‌اند.



شکل ۳: سرویس‌های نامزد و برنامه‌ی اجرا

پس از مشخص‌شدن لیست سرویس‌های هر عمل، از برخی از آن‌ها گره‌ای ساخته و آن‌ها را با یالی جهت‌دار به‌هم ارتباط می‌دهیم. یال‌ها به صورت خطوطی جهت‌دار بین سرویس‌های مجاور در شکل ترسیم شده و وزن هر یال کنار آن نوشته شده است.

عملیات ساخت گراف توسط یکی اجزای سیستم صورت می‌گیرد که به طور دقیق در الگوریتم ۱ معرفی شده است. در این الگوریتم، ابتدا به ازای هر عمل، کلیه‌ی سرویس‌ها از واسطه دریافت می‌شود. سپس این سرویس‌ها در لیستی از مجموعه‌ی سرویس‌های هم‌انجمن به‌نام Now قرار می‌گیرد. در مرحله‌ی بعدی برای هر انجمن در لیست بررسی می‌شود که آیا سرویس

[27] static
[28] dynamic

هم‌خوانی در لیست عمل قبلی به‌نام Past، می‌یابد یا نه. اگر برای هر سرویس در لیست عمل قبلی آن، سرویسی هم‌خوان وجود داشته باشد (مثل انجمن سرویس s31 در شکل ۳)، به‌صورت دوبه‌دو یال‌هایی با وزن کیفیت سرویس مقصد بین سرویس‌های آن‌دو انجمن در نظر می‌گیرد (یال‌های s21_s31 و s22_s31). سپس آن انجمن (انجمن حاوی s21 و s22) را از لیست عمل قبلی حذف کرده تا دیگر مورد جستجو قرار نگیرد. اما اگر برای هر سرویس در لیست عمل قبلی آن، هیچ انجمن همسانی را نیافت (مثل انجمن حاوی سرویس‌های s42 و s43 در شکل ۳)، آن انجمن را از لیست فعلی حذف کرده تا هنگام بررسی عمل بعدی، مورد جستجو قرار نگیرد. این حذفیات باعث می‌شود از جستجوهای اضافه برای یافتن سرویس‌های مناسب پرهیز شده و در نتیجه سرعت ساخت گراف افزایش یابد.

```
Past=NULL; Now=NULL;
create_node(nb,B); create_node(nf,E);
foreach n in Tasks do{
   Past=Now;  Now= get_cm(TRn);
   foreach CM in Now do{
      if n=1 then CMP= {B}
      else CMP= match(CM,Past);
      if CMP is not NULL then{
         foreach s in CM do{
            create_node(ns,s);
            foreach sp in CMP do
               add_edge(nsp,ns,Q(s));
            if n=k-1 then
               add_edge(ns,nf,0);
         }
         remove(CMP,Past);
      } else
         remove(CM,Now);
    }
   if Now is NULL then
      throw("It is not feasible!");
}
```

الگوریتم ۱: ساخت گراف

پس در ابتدای اجرای الگوریتم برای عمل nام، Now حاوی کلیه‌ی سرویس‌های متناظر آن عمل است درحالی‌که Past شامل سرویس‌هایی ازعمل n-1ام می‌باشد که سرویس هم‌خوانی در مجموعه سرویس‌های عمل n-2ام یافته‌اند. مثلا اگر در ابتدای اجرای الگوریتم برای عمل t3 باشیم، لیست Now حاوی سرویس‌های s31, s32, s33 است و سرویس‌های s21, s22, s23 در لیست Past قرار دارند. در این نقطه از اجرا، سرویس‌های مربوط به اعمال پس از t3، هنوز مشخص نشده است.

بنابراین از سرویس‌هایی که در لیست عمل قبلی، سرویس هم‌خوانی ندارند گره‌ای نیز ساخته نمی‌شود و حافظه‌ی اضافی مصرف نمی‌گردد. گراف نهایی در شکل ۳، شامل کلیه‌ی سرویس‌ها و یال‌های رسم شده به‌جز سرویس‌های s24، s32، s42 وs43 می‌باشد.

ممکن است سرویس‌های موجود در فضای سیستم هم‌خوان نبوده و نتوان سرویس مرکبی از آن‌ها ساخت. در این حالت هنگامی‌که لیست Now در پایان اجرای الگوریتم برای یکی از اعمال، خالی شد، می‌توان عملی[29] نبودن ترکیب را با پایان دادن اجرای الگوریتم همراه با پیامی مناسب، اعلام داشت.

در الگوریتم ۱ هر تابع با عملیات زیر تعریف شده است:

- `create_node(nodeName,service)`: یک گره متناظر با service می‌سازد و نام آن را nodeName می‌گذارد.
- `get_cm(TRn)`: درخواست کلیه‌ی سرویس‌های موجود با کارکرد منطبق با عمل nام را از واسطه می‌نماید. واسطه آن سرویس‌ها را در چند مجموعه از سرویس‌های هم‌انجمن قرار داده و به درخواست‌کننده تحویل می‌دهد.
- `match(CM,Past)`: در لیستی به‌نام Past جستجو می‌کند و اگر انجمنی یکسان با انجمن CM یافت، آن را برمی‌گرداند در غیر این‌صورت NULL را به عنوان خروجی اعلام می‌دارد.
- `add_edge(node1,node2,Q)`: یالی جهت‌دار از گره‌ی node1 به گره‌ی node2 متصل نموده و Q را به عنوان وزن آن یال قرار می‌دهد.

[29] feasibility

- `remove(community,list)` : انجمن `community` را از لیست `list` حذف می‌نماید.
- `throw(message)` : پایان‌دادن به اجرای الگوریتم به‌همراه صدور پیام `message`.

پس از اجرای الگوریتم ۱، گرافی ایجاد می‌شود با مسیرهای مختلف که فقط تعدادی از آن‌ها مسیری از سرویس متناظر با عمل ابتدایی به سرویس متناظر با عمل انتهایی (مسیر اجرا[30]) را می‌سازد. مسیرهای اجرا گره‌ی آغازین گراف را به گره‌ی پایانی آن (در شکل ۳، به‌ترتیب b و e) مرتبط می‌سازند.

مجموعه سرویس‌های هر یک از مسیرهای اجرا در گراف را می‌توان به‌عنوان سرویس مرکب درنظر گرفت ولی باید یکی از مسیرهای اجرا را به‌عنوان بهترین ترکیب برگزینیم. این کار را با جستجو در گراف به‌منظور یافتن بهترین مسیر انجام می‌دهیم.

۳-۲- جستجو

کیفیت کلی سرویس مرکب برابر مجموع کیفیت سرویس‌های سازنده‌ی آن است و برای یافتن بهترین ترکیب باید مجموعه‌ای از سرویس‌ها انتخاب شود که بیشترین مقدار این مجموع را دارد. به‌خاطر این‌که مسیرهای دارای یال‌های سنگین‌تر از سرویس‌های با کیفیت‌تری برخوردارند، برای یافتن بهترین و باکیفیت‌ترین سرویس مرکب، سنگین‌ترین مسیر را از بین مسیرهای اجرا برمی‌گزینیم.

پس برای یافتن بهترین سرویس مرکب، بهترین مسیر را در گراف ساخته شده می‌یابیم. بدین منظور جستجویی را در گراف انجام می‌دهیم. از بین روش‌های مختلفی که تابه‌حال برای جستجوی سریع در گراف بیان شده است، روش برنامه‌نویسی پویا توانسته است با پیچیدگی مطلوبی به جستجوی بهترین مسیر در گراف بپردازد [۱۳].

پس از جستجو در گراف شکل ۳، مسیر $\{b, s13, s23, s33, s44, e\}$ به‌عنوان مسیر اجرا معرفی می‌گردد.

۳-۳- ارزیابی

وجود تامین‌کنندگان متعدد باعث ایجاد سرویس‌های همکار با واسط‌های مختلف می‌گردد. اگر به این گوناگونی واسط‌ها در دنیای واقعی توجهی نشود گراف حاصله از تعداد زیادی گره‌ی اضافی تشکیل می‌گردد که این امر باعث تولید زمان‌گیر گرافی حجیم می‌شود و در ضمن جستجوی چنین گرافی نیز زمان بیشتری طلب می‌کند. الگوریتم ۱ با توجه به این نکته، توانسته است به‌میزان قابل توجهی از حافظه‌ی مصرفی بکاهد و با سرعت بیشتری دو عمل ساخت و جستجو را انجام داده تا کاربر سریع‌تر به سرویس مورد نظر خود دست یابد. علاوه بر آن، این الگوریتم به محض فهمیدن عملی نبودن ترکیب از مجموعه سرویس‌های موجود، اجرا را خاتمه داده و از اتلاف منابع جلوگیری می‌کند.

۴- پیاده‌سازی

الگوریتم ۱ را بر سیستمی با پردازشگر Intel Pentium© IV[31] و حافظه‌ی اصلی ۲۵۶ مگابایتی توسط زبان برنامه‌نویسی C# پیاده‌سازی می‌نماییم. سپس در سیستم‌عامل ویندوز XP، آن‌را اجرا کرده و سرویس‌هایی با مشخصه‌های تصادفی را به‌عنوان ورودی آن در نظر می‌گیریم.

الگوریتم ارائه شده توانسته است بهبودهایی را در زمینه‌ی سرعت اجرا و حجم مورد نیاز نسبت به کارهای قبلی (مثل [۱۳] و [۱۴]) ایجاد کند. برای ارزیابی تجربی، روش معرفی‌شده را با آخرین کار صورت‌گرفته در این زمینه [۱۳]، پس از پیاده‌سازی، مقایسه می‌کنیم. یعنی در اجرای هر دو، با ورودی‌های یکسان، زمان و حافظه‌ی مصرفی را محاسبه می‌کنیم.

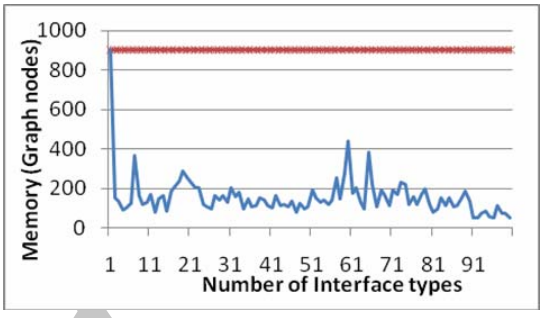
در برخورد با مسائل مختلف، برنامه‌های اجرای مختلفی را با اعمال متفاوت شاهد هستیم و با توجه به

[30] execution path

[31] ویژگی‌های Celeron® D : 2.4 GH, 32 bit, 256 KB Cache memory

سیستم، تعداد سرویس‌های با عملکرد منطبق بر هر عمل در حال تغییر است. این مجموعه نیز در اجراهای مختلف با واسط‌های مختلفی یافته می‌شوند.

در نتیجه باید در بررسی‌ها و مقایسه‌های میزان مصرف زمان و حافظه، به تغییرات[32] تعداد اعمال، تعداد سرویس‌ها و گوناگونی سرویس‌ها در خلال زمان توجه شود. زمان اجرای الگوریتم، مصرف زمان و تعداد گره‌های گراف حافظه‌ی مصرفی را نشان می‌دهد. این دو مقدار را در اجراهای مختلف با در نظر گرفتن یکی از تغییرات به‌دست آورده و در قالب نمودارهای شکل ۵ تا شکل ۷(الف)

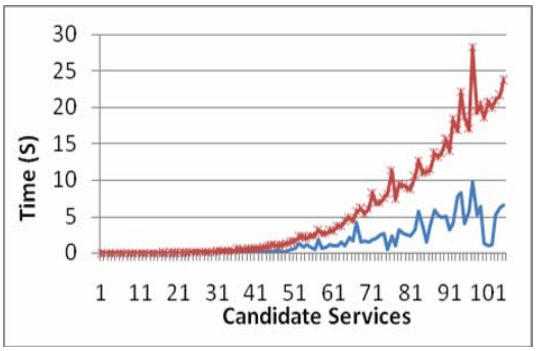


(ب)

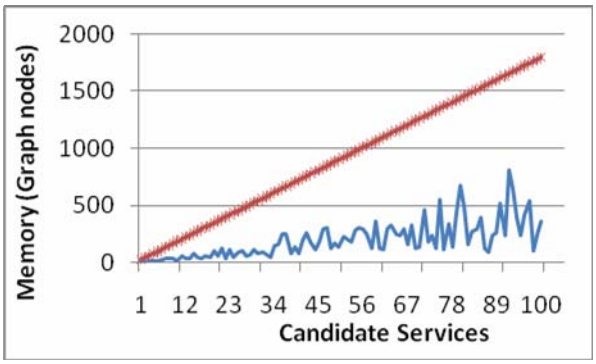
شکل نمایش می‌دهیم. نمودار «الف» از هر شکل بیانگر زمان اجرای الگوریتم‌هاست درحالی‌که نمودار «ب»، میزان حافظه‌ی مورد نیاز هر الگوریتم را نشان می‌دهد. در هر نمودار منحنی بالایی مربوط به کارهای پیشین و پایینی مربوط به روش پیشنهاد شده در این نوشتار است (شکل ۴).

شکل۴: راهنمای نمودارها

در شکل ۵ برنامه‌ی اجرایی با بیست عمل در نظرگرفته و تعداد سرویس‌های نامزد را متغیر می‌گیریم. واسط هر سرویس از بین ده واسط، به‌صورت تصادفی انتخاب می‌شود.



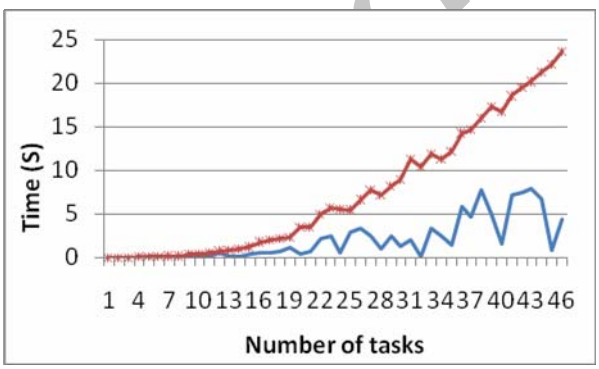
(الف)



(ب)

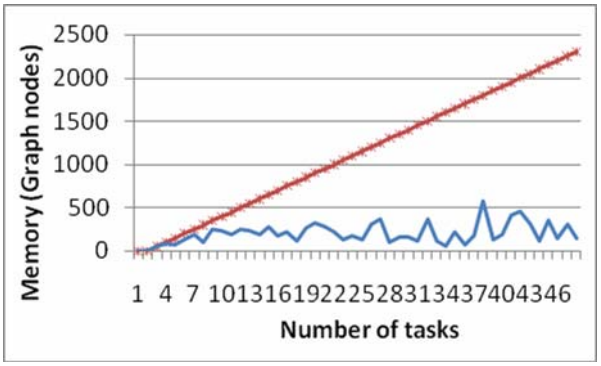
شکل۵: تغییر در تعداد سرویس‌های نامزد

در شکل ۶ تعداد عمل‌های برنامه‌ی اجرا متغیر است. برای هر عمل پنجاه سرویس نامزد با واسطی از بین ده واسط موجود در نظر گرفته شده است.



(الف)

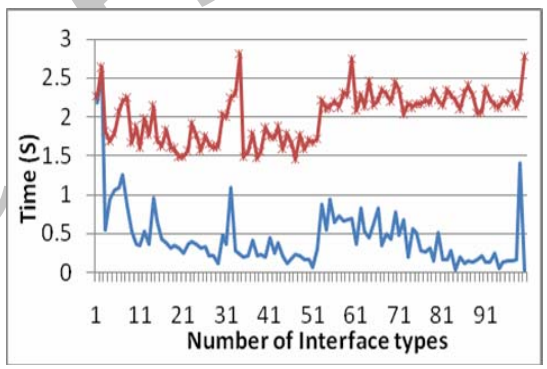
[32] varies



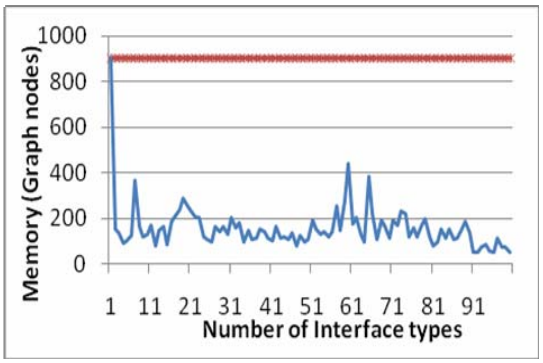
(ب)

شکل۶: تغییر در تعداد اعمال برنامه‌ی اجرا

در شکل ۷شکل برنامه‌ی اجرایی با بیست عمل وپنجاه سرویس نامزد در نظر گرفته شده است. واسط هر سرویس از بین تعداد متغیری از واسط‌ها، انتخاب می‌شود.



(الف)



(ب)

شکل۷: تغییر در تعداد گوناگونی واسط‌ها

۵- جمع‌بندی

در ابتدا مختصری با سیستم‌های سرویس‌گرا آشنا شدیم و دیدیم که برای رفع نیازهای کاربر در اکثر اوقات، باید سرویس مرکبی از سرویس‌های موجود در محیط سیستم بسازیم. نوع سرویس‌های شرکت‌کننده در ترکیب و توالی آن‌ها در برنامه‌ی اجرایی که در دست داریم، مشخص می‌شود.

به‌خاطر تغییراتی که در کیفیت سرویس‌ها و تامین‌کنندگان آن‌ها در طول زمان به‌وجود می‌آید بهتر است عمل ترکیب در زمان اجرا و به‌صورت پویا انجام شود. وجود تامین‌کنندگان مختلف و گوناگونی در واسط سرویس‌ها باعث می‌گردد که هر دو سرویسی را نتوان با یکدیگر ترکیب کرد.

در ادامه الگوریتمی ارائه شد که چگونگی ترکیب سرویس‌ها به صورت پویا را بیان می‌داشت. در این الگوریتم طبق برنامه‌ی اجرا ابتدا گرافی از سرویس‌های نامزد ساخته می‌شد (با توجه به هم‌خوانی واسط‌ها) سپس بهترین مسیر در گراف به‌عنوان ترکیب بهینه معرفی می‌شود.

از لحاظ تجربی نیز دیده شد که این الگوریتم نسبت به روش‌های قبلی از سرعت بهتر و مصرف حافظه‌ی کمتری برخوردار است و می‌تواند در هر نقطه از اجرا عملی نبودن ترکیب با سرویس‌های موجود را تشخیص دهد.

۶- منابع

[1] Mike P., Papazoglou and Willem-Jan van den, Heuvel., "Service oriented architectures approaches, technologies and research issues." s.l. : The VLDB, Springer-Verlag, 2007, Issue 16, pp. 389-415.

[2] papazoglou, et al., "service-oriented computing: state of the art and research challenges." s.l. : IEEE Computer society, Nov. 2007, pp. 38-45.

[3] Knoblock, Craig A., "Building Software Agents for Planning, Monitoring, and Optimizing Travel." s.l. : ENTER, 2004.

[4] Budak, Arpinar, et al., "Ontology-Driven Web Services Composition Platform." s.l. : IEEE International Conference on E-Commerce Technology (CEC'04), 2004.

[5] L., Zeng, et al., "Quality Driven Web Services Composition." s.l. : 12th Int'l Conf,World Wide Web(WWW), 2003.

[6] Jinghai, Rao and Xiaomeng, Su., "A Survey of Automated Web Service Composition Methods." s.l. : ICAPS, 2005.

[7] Biplav, Srivastava and Jana, Koehler., "Web Service Composition - Current Solutions and Open Problems." s.l. : ICAPS, 2004.

[8] Benatallah, B., Sheng, Q. Z. and Dumas, M., "The Self-Serv Environment for Web Services Composition." s.l. : IEEE Internet Computing, 2003, Issue 1, Vol. 7, pp. 40 - 48.

[9] Ambite, J. L., et al., "Getting from Here to There: Interactive Planning and Agent Execution for Optimizing Travel." s.l. : Proceedings of the Fourteenth Conference on Innovative Applications of Artificial Intelligence (IAAI-2002), 2002.

[10] Zeng, Liangzhao, et al., "QoS-Aware Middleware for Web Services Composition." s.l. : IEEE Transactions on Software Engineering, 2004, Issue 5, Vol. 30, pp. 311-327.

[11] Yu, Tao and Lin, Kwei-Jay., "Service Selection Algorithms for Web Services with End-to-end QoS Constraints." s.l. : Journal of Information Systems and e-Business Management, 2005, Issue 2, Vol. 3, pp. 103-126.

[12] Fluegge, Matthias, et al., "Challenges and Techniques on the Road to Dynamically Compose Web Services." s.l. : ICWE'06, ACM, 2007.

[13] Gao, Yan, et al., "Optimal Web Services Selection Using Dynamic Programming." s.l. : Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06) , 2006.

[14] Gao, Yan, et al., "Optimal Selection of Web Services for Composition Based on Interface-Matching and Weighted Multistage Graph." s.l. : Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05) IEEE, 2005.

[15] Sheshagiri, Mithun, desJardins, Marie and Finin, Timothy., "A Planner for Composing Services Described in DAML-S." s.l. : ICAPS, 2003.

[16] Sirin, Evren and Parsia, Bijan., "Planning for Semantic Web Services." s.l. : Semantic Web Conference (ISWC), 2004.