





دانشگاه آزاد اسلامی واحد سیرجان
دانشکده فنی و مهندسی گروه نرم افزار

پایان نامه برای دریافت درجه کارشناسی ارشد M.Sc

گرایش نرم افزار

بیان روشی به منظور تخصیص ماشین های مجازی برای
جلوگیری از سر بار شدن میزبان های فیزیکی با هدف بهبود
کیفیت سرویس در مراکز داده ابری

استاد راهنما

دکتر محمد صادق حاج محمدی

نگارش

صائب ملایی ندیکی

تابستان ۱۳۹۶

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه

نام دانشکده: دانشکده فنی و مهندسی

نام دانشجو: صائب ملایی ندیکی

عنوان پایان‌نامه: بیان روشی به منظور تخصیص ماشین های مجازی برای جلوگیری از سربار شدن

میزبان های فیزیکی با هدف بهبود کیفیت سرویس در مراکز داده ابری

تاریخ دفاع: تابستان ۱۳۹۶

رشته: مهندسی کامپیوتر

گرایش: نرم افزار

ردیف	سمت	نام و نام خانوادگی	دانشگاه یا مؤسسه	امضاء
۱	استاد راهنما	دکتر محمدصادق حاج محمدی	دانشگاه آزاد اسلامی سیرجان	
۲	استاد مدعو	دکتر خطیبی	دانشگاه آزاد اسلامی سیرجان	
۳	استاد مدعو	دکتر نورمندی	دانشگاه آزاد اسلامی سیرجان	



معاونت پژوهش و فناوری

به نام خدا

منشور اخلاق پژوهش

بیامری از خداوند سبحان و اعتقاد به این که عالم محضر خداست و بهواره ناظر بر اعمال انسان و به منظور پاس داشت مقام بلند دانش و پژوهش و نظریه ایست جایگاه دانشگاه در اعتلای فرهنگ و

تمدن بشری، مادر انجمن و اعضاء بیات علمی و احدای دانشگاه آزاد اسلامی متعهد می گردیم اصول زیر را در انجام فعالیت های پژوهشی مد نظر قرار داده و از آن تخطی نکنیم:

۱. اصل حقیقت جویی: تلاش در راستای پی جویی حقیقت و وفاداری به آن و دوری از حرکت پنهان سازی حقیقت.
۲. اصل رعایت حقوق: الزام به رعایت کامل حقوق پژوهشگران و پژوهیدگان (انسان، حیوان و نبات) و سایر صاحبان حق.
۳. اصل مالکیت مادی و معنوی: تعهد به رعایت کامل حقوق مادی و معنوی دانشگاه و کلیه بکاران پژوهش.
۴. اصل منافع ملی: تعهد به رعایت مصالح ملی و در نظر داشتن پیشبرد توسعه کشور در کلیه مراحل پژوهش.
۵. اصل رعایت انصاف و امانت: تعهد به اجتناب از حرکت جانب داری غیر علمی و حفاظت از اموال، تجهیزات و منافع در اختیار.
۶. اصل رازداری: تعهد به صیانت از اسرار و اطلاعات محرمانه افراد، سازمان ها و کشور و کلیه افراد و نهادهای مرتبط با تحقیق.
۷. اصل احترام: تعهد به رعایت حریم ها و حرمت ها در انجام تحقیقات و رعایت جانب تقد و خودداری از حرکت حرمت شکنی.
۸. اصل ترویج: تعهد به رواج دانش و اشاعه نتایج تحقیقات و انتقال آن به بکاران علمی و دانشجویان به غیر از مواردی که منع قانونی دارد.
۹. اصل برائت: الزام به برائت جویی از حرکت رفتار غیر حرفه ای و اعلام موضع نسبت به کسانی که حوزه علم و پژوهش را به شبهه های غیر علمی می آلاینند.



دانشگاه آزاد اسلامی واحد شیراز

حوزه معاونت، پژوهش و فناوری

تأییدیه‌ی صحت و اصالت نتایج

اینجانب صائب ملایی ندیکی دانش آموخته مقطع کارشناسی ارشد M.Sc در رشته مهندسی کامپیوتر که در تاریخ ۱۳۹۶/۰۶/۲۶ از پایان نامه خود تحت عنوان ”بیان روشی به منظور تخصیص ماشین های مجازی برای جلوگیری از سرشار شدن میزبان های فیزیکی با هدف بهبود کیفیت سرویس در مراکز داده ابری” با کسب نمره و درجه دفاع نموده ام بدینوسیله متعهد می شوم:

۱. این پایان نامه حاصل تحقیق و پژوهش انجام شده توسط اینجانب بوده و در مواردی که از دستاوردهای علمی و پژوهشی دیگران (اعم از پایان نامه، کتاب، مقاله و) استفاده نمودهام، مطابق ضوابط و رویه موجود، نام منبع مورد استفاده و سایر مشخصات آن را در فهرست مربوطه ذکر و درج کرده ام.

۲. این پایان نامه قبلاً برای دریافت هیچ مدرک تحصیلی (هم سطح، پایین تر یا بالاتر) در سایر دانشگاهها و موسسات آموزش عالی ارائه نشده است.

۳. چنانچه بعد از فراغت از تحصیل، قصد استفاده و هر گونه بهره برداری اعم از چاپ کتاب، ثبت اختراع و ... از این پایان نامه داشته باشم، از حوزه معاونت پژوهشی واحد مجوزهای مربوطه را اخذ نمایم.

۴. چنانچه در هر مقطع زمانی خلاف موارد فوق ثابت شود، عواقب ناشی از آن را می پذیرم و واحد دانشگاهی مجاز است با اینجانب مطابق ضوابط و مقررات رفتار نموده و در صورت ابطال مدرک تحصیلیام هیچگونه ادعایی نخواهم داشت.

نام و نام خانوادگی: صائب ملایی ندیکی

تاریخ و امضا:

قدردانی

در آغاز وظیفه خود می‌دانم از زحمات بی‌دریغ استاد راهنمای خود، جناب آقای دکتر جاج‌محمدی صمیمانه تشکر و قدردانی کنم که قطعاً بدون راهنمایی‌های ارزنده ایشان، این مجموعه به انجام نمی‌رسید. همچنین لازم می‌دانم از فعالان حوزه نرم‌افزار آزاد که بدون هیچ چشم‌داشتی پاسخ همه پرسش‌های علاقه‌مندان به این حوزه را می‌دهند و منابع فعالیت‌های خود را در اختیار همه‌گان می‌گذارند. در پایان از پدیدآورندگان بسته زی‌پرشین، مخصوصاً جناب آقای وفا خلیقی، که این پایان‌نامه با استفاده از این بسته، آماده شده است و همه دوستانمان در گروه پارسی‌لاتک کمال قدردانی را داشته باشم.

صائب ملایی ندیکی

تابستان ۱۳۹۶

تقديم به :

الكساندرا الباكيان

که با شعار ”برای حذف همه موانع از راه علم“

To remove all barriers in the way of science

۵۰ میلیون مقاله را مجانی در اختیار دنیا قرار داد

فهرست مطالب

چکیده	۱
-------	---

فصل اول: مقدمه

۱-۱ مقدمه ای بر رایانش ابری	۳
۲-۱ لایه ها و سرویس های سیستم های رایانش ابری	۳
۳-۱ مجازی سازی منابع و مفهوم ترکیب در سیستم های ابری	۵
۴-۱ پرسش اصلی	۶
۵-۱ تعریف مساله	۷
۶-۱ اهداف تحقیق به صورت کلی و جزئی	۷
۷-۱ فرضیه های تحقیق	۷
۸-۱ ابزارهای اندازه گیری	۸
۹-۱ جنبه نوآوری و جدید بودن تحقیق در چیست	۸
۱۰-۱ مراحل پایان نامه	۸

فصل دوم: مروری بر روش های انجام شده

۱-۲ مروری بر روش های انجام شده	۱۰
۲-۲ روش چوی	۱۰
۳-۲ روش چین	۱۱
۴-۲ روش باسکار	۱۱
۵-۲ گودرزی	۱۲
۶-۲ روش اسماعیل	۱۳
۷-۲ روش راجو	۱۴
۸-۲ روش دان	۱۴

۹-۲	روش پاتل	۱۵
-----	----------	----

فصل سوم: روش پیشنهادی

۱-۳	مقدمه	۱۹
۲-۳	تجزیه و تحلیل روش پیشنهادی	۱۹
۳-۳	روش ترکیب ماشین‌های مجازی	۲۰
۴-۳	بدست آوردن میزان انرژی مصرفی	۲۲
۵-۳	بدست آوردن میزان نقض کیفیت سرویس	۲۳

فصل چهارم: بررسی و ارزیابی راه حل پیشنهادی

۴-۱	محیط آزمایش	۲۶
۴-۲	نتایج مربوط به شبیه سازی	۲۷

فصل پنجم: جمع بندی و کارهای آینده

۵-۱	جمع بندی و کارهای آینده	۳۴
-----	-------------------------	----

مراجع

۳۶	مراجع
----	-------

پیوست‌ها

۳۹	پیوست‌ها
----	----------

فهرست جداول

۱۷	۱-۲ مقایسه مقالات بر اساس سه معیار
۲۷	۲-۴ مشخصات ماشین‌های مجازی
۲۷	۳-۴ مشخصات میزبان فیزیکی

فهرست نمودارها

۱-۴	مقایسه مصرف انرژی با سیاست MMT	۲۸
۲-۴	مقایسه مصرف انرژی با سیاست MU	۲۹
۳-۴	تعداد مهاجرت‌های رخ داده با سیاست MMT	۳۰
۴-۴	تعداد مهاجرت‌های رخ داده با سیاست MU	۳۰
۵-۴	مقایسه کیفیت سرویس نقض شده در روش پیشنهادی و مورد مقایسه با سیاست MMT	۳۱
۶-۴	مقایسه کیفیت سرویس نقض شده در روش پیشنهادی و مورد مقایسه با سیاست MU	۳۲

فهرست تصاویر

۱-۱ لایه‌های سرویس ابری ۴

چکیده

امروزه با پیشرفت روز افزون فناوری اطلاعات و افزایش برنامه های کاربردی، بی شک نیاز به محاسبات یکپارچه برای کاربران ضروری می باشد. بنابراین استفاده از تکنولوژی مانند رایانش ابری که با توجه به نیاز کاربران، پردازش های محاسباتی آن ها را انجام دهد و نتایج را به آن ها نمایش دهد، لازم می باشد. در حال حاضر چالش های متنوعی در زمینه رایانش ابری مطرح است. استفاده مؤثر، تخصیص و مدیریت منابع به منظور بهبود کیفیت سرویس و بهره‌وری انرژی یکی از از چالش های مهم در سیستم های ابری است. در این پایان نامه تمرکز بر روی انتخاب مقصد مناسب برای میزبانی ماشین های مجازی و همچنین اعمال کنترلی در مهاجرت ماشین های مجازی مهاجر می باشد. هدف ما از انجام این کار این است که تخصیص ماشین های مجازی به میزبان های فیزیکی را به چه نحوی انجام دهیم که تا جای ممکن از سریز شدن میزبان های فیزیکی و مهاجرت اضافی جلوگیری کنیم. نتایج تجربی آزمایشات در مقایسه با روش پایه نشان دهنده این است که با تخصیص مناسب و اعمال کنترلی در مهاجرت می توانیم در بهبود کیفیت سرویس تأثیرگذار باشیم درحالی که از افزایش مصرف انرژی جلوگیری می کنیم.

واژگان کلیدی: رایانش ابری، تخصیص، سرریز شدن، کنترل مهاجرت، کیفیت سرویس، مصرف انرژی.

فصل اول : مقدمه

۱-۱- مقدمه ای بر رایانش ابری

امروزه با پیشرفت روز افزون فناوری اطلاعات و افزایش برنامه‌های کاربردی، بی‌شک نیاز به محاسبات مسنجم و یکپارچه برای کاربران ضروری می‌باشد. همچنین با توجه به نیازهای کاربردی که کاربران دارند، نیاز است که کاربران بتوانند کارهای پیچیده خود را بدون اینکه نیازی به داشتن سخت افزارها و نرم افزارهای گران قیمت داشته باشند، از طریق اینترنت بتوانند انجام دهند. در واقع با این پردازش‌های سخت و سنگین، نیاز به پردازنده‌های متنوع و زیاد دارند تا بتوانند این کارهای پیچیده را با آنها انجام دهند. بنابراین استفاده از تکنولوژی مانند رایانش ابری که با توجه به نیاز کاربران، پردازش‌های محاسباتی آنها را انجام دهد و نتایج را به آنها نمایش دهد، لازم می‌باشد. سیستم‌های رایانش ابری مراکز داده را با طراحی به صورت شبکه‌های مجازی، از نظر سخت افزار، پایگاه داده، نرم افزار و ... توانمند کردند، به طوری که کاربران بتوانند برنامه‌های کاربردی و مورد نیاز خود را از هر جایی با کمترین هزینه دریافت کنند. (مل و گرنس^۱ ۲۰۱۱، ریتینگ‌هاوس و رنسام^۲ ۲۰۱۶)

انجمن ملی استانداردها و تکنولوژی سیستم‌های رایانش ابری را اینگونه تعریف می‌کند: سیستم‌های رایانش ابری مدلی برای فراهم کردن دسترسی آسان بر طبق نیاز کاربران به مجموعه ای از منابع که قابل تغییر از طریق اینترنت هستند، می‌باشد. (مل و گرنس^۱ ۲۰۱۱)

۱-۲- لایه‌ها و سرویس‌های سیستم‌های رایانش ابری

سیستم‌های رایانش ابری از مجموعه ای از لایه‌ها تشکیل شده است که برنامه‌های کاربران بر روی این لایه‌ها نصب و اجرا می‌گردد. این لایه‌ها در سه سطح متفاوت به نام‌های زیرساخت تحت یک سرویس^۳ (IaaS)، پلت فرم تحت یک سرویس^۴ (PaaS) و نرم افزار تحت یک سرویس^۵ (SaaS) ارائه می‌شوند. در زیر به معرفی هر سرویس می‌پردازیم:

۱. سطح اول که با IaaS شناخته می‌شود، سرویس‌های زیرساخت ابری نام دارد که سیستمی را که عموماً به صورت یک بستر مجازی سازی شده می‌باشد را به صورت سرویس ارائه می‌دهند. در

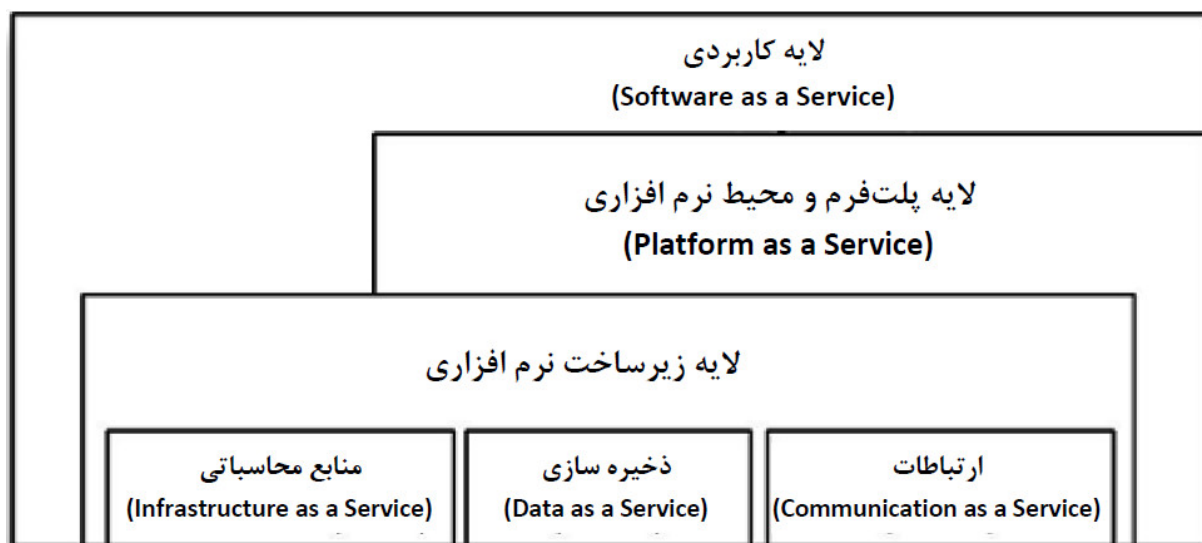
¹ Mel & Grance

² Rittinghouse & Ransome

³ Infrastructure as a Service

⁴ Platform as a Service

⁵ Software as a Service



شکل ۱-۱ لایه‌های سرویس ابری

این سطح، کاربران به جای خرید سخت‌افزار، نرم‌افزار و تجهیزات شبکه، تمام این امکانات و زیر ساخت‌ها را به صورت یک سرویس مجازی خریداری می‌کنند. در واقع تجهیزات مورد نیاز براساس یک مدل که بر پایه قیمت گذاری براساس استفاده آن‌ها از منابع می‌باشد، ارائه می‌شود. از آنجا که این منبع ممکن است تغییر کند، این چارچوب هم به صورت پویا براساس نیاز به منابع تغییر می‌کند. نمونه ارائه کننده این سرویس‌ها مانند شرکت آمازون می‌باشد.

۲. در سطح بعدی که با PaaS نمایش داده می‌شود، محیطی برای تولید برنامه‌ها و همچنین تست آن‌ها را فراهم می‌آورد.

۳. در سطح بعدی که با SaaS نمایش داده می‌شود، در واقع این سطح نرم‌افزاری است که از طریق اینترنت و براساس الگوی قیمت گذاری مشخص شده براساس مصرف کاربر در اختیار آن‌ها قرار داده می‌شود. برای نمونه می‌توانیم به گوگل داک در سایت گوگل اشاره کرد (هافر و کرگیانس^۶ ۲۰۱۱)

⁶ Höfer & Karagiannis

۱-۳- مجازی سازی منابع و مفهوم ترکیب در سیستم های ابری

مجازی سازی سطح جدیدی از انعطاف پذیری را برای استفاده از منابع ماشین های فیزیکی^۷ (PM) فراهم می کند و امکان یکپارچه سازی منابع فیزیکی در قالب منابع مجازی را ایجاد می کند. در محیط سیستم های رایانش ابری از تکنیک مجازی سازی استفاده می شود. تکنیک مجازی سازی این امکان را فراهم می کند که چندین نرم افزار که در واقع روی ماشین های مجازی^۸ (VM) قرار داده می شوند را همزمان بر روی تنها یک کامپیوتر اجرا کنیم از جمله مهم ترین اهداف مجازی سازی می توانیم به موارد زیر اشاره کنیم.

• بهره وری و بهینه سازی در استفاده از منابع

با ویژگی مجازی سازی، ماشین های مجازی می توانند یکپارچه شوند و به سیستم های بیکار یا در حال استفاده فرستاده شوند. با استفاده از مجازی سازی، سیستم های موجود می توانند یکپارچه شوند. در واقع مجازی سازی یک فرصت برای یکپارچه سازی و بهینه سازی معماری سیستم ها، زیرساخت برنامه ها، پایگاه های داده، را فراهم می آورد که کارایی بالاتر را نتیجه می دهد.

• کمتر مصرف کردن برق و در نتیجه کاهش هزینه ها استفاده از مجازی سازی این امکان را فراهم می آورد که میزان انرژی مصرفی کاهش یابد و در هزینه ها و سرمایه های استفاده شده به طور قابل توجهی صرفه جویی به عمل آید.

• صرفه جویی شدن در فضا بزرگ بودن و جاگیر بودن سرورهای فیزیکی یک مساله بزرگ در مراکز داده ابری می باشد. مجازی سازی می تواند این مشکل را با یکپارچه کردن تعداد زیادی ماشین های مجازی بر روی تعداد کمی میزبان های فیزیکی بر طرف کند.

در سیستم های مجازی سازی ما از مفاهیمی مانند ماشین مجازی، ماشین فیزیکی، مهاجرت و ترکیب^۹ استفاده می کنیم. طبق بیانات قبلی، ماشین مجازی مانند یک سیستم واقعی است که بر روی این ماشین می توانیم نرم افزارها و یا سیستم عامل های مورد نیاز کاربران را نصب کنیم. بعد از نصب نرم افزارها و یا سیستم عامل های مورد نظر روی این ماشین های مجازی، در نهایت این ماشین های مجازی بر روی یک ماشین فیزیکی که در واقع یک سرور کامپیوتری با قابلیت های بالایی است، اجرا می شود. هر ماشین فیزیکی می تواند به طور همزمان چندین ماشین مجازی با نیاز به منابع متفاوت را بر روی خودش اجرا کند. بین ماشین های فیزیکی زمانی که یک ماشین فیزیکی بار زیادی روی آن قرار بگیرد و منابع لازم را برای

⁷ Physical machine

⁸ Virtual machine

⁹ Consolidation

ماشین مجازی نداشته باشد، از امکانی به نام مهاجرت در بین ماشین‌های فیزیکی می‌توانیم استفاده کنیم. در واقع از این امکان برای انتقال ماشین‌های مجازی برای اینکه بتوانیم از منابع ماشین‌های فیزیکی به گونه‌ای مناسب استفاده کنیم، استفاده می‌شود. با مهاجرت ماشین‌های مجازی می‌توانیم ماشین‌های مجازی را تا جای ممکن که آن ماشین فیزیکی ظرفیت دارد بر روی آن قرار دهیم و از منابع ماشین فیزیکی حداکثر استفاده را بکنیم و ماشین‌های فیزیکی اضافه را خاموش کنیم، به این عمل ترکیب گفته می‌شود. در واقع این روش تکنیک موثری است که با بالا بردن بهره‌وری از منابع و حداقل کردن تعداد ماشین‌های فیزیکی مصرف انرژی را کاهش می‌دهد. این تکنیک با مهاجرت ماشین‌های مجازی از روی ماشین‌های فیزیکی بیکار به میزبان‌های دیگر و سپس تغییر وضعیت ماشین‌های فیزیکی بیکار به حالت خواب سعی دارد مصرف انرژی را کاهش دهد و از منابع به طور موثری استفاده کند. (گارسیاوالاس و لو^{۱۰} ۲۰۱۴، ۶۰)

اگرچه ترکیب پویای ماشین‌های مجازی ممکن است کارایی مراکز داده را بهبود بخشد، اما به دلیل قرار گرفتن چندین ماشین مجازی روی یک ماشین فیزیکی، تضمین کردن سرویس‌های مورد نظر به کاربران یکی از چالش‌های بزرگ مربوط به این تکنیک می‌باشد. کیفیت سرویس مربوط به کاربران معمولاً با توافق نامه سطح خدمات^{۱۱} ارائه می‌شود (سوبرامانیان و پاتابی^{۱۲} ۲۰۱۶)

ترکیب بهینه ماشین‌های مجازی شامل سه بخش می‌باشد:

۱. شناسایی ماشین‌های فیزیکی سربار شده

۲. شناسایی ماشین‌های فیزیکی کم‌بار شده

۳. انتخاب ماشین‌های مجازی برای مهاجرت از ماشین‌های سربار

۴-۱- پرسش اصلی

در این رساله قصد داریم به این پرسش پاسخ دهیم که به چه نحوی عمل جایابی ماشین‌های مجازی را به میزبان‌های فیزیکی انجام دهیم تا از منابع میزبان‌های فیزیکی به گونه‌ای مناسب استفاده کنیم تا بتوانیم در بهبود کیفیت سرویس و توان مصرفی موثر واقع شویم.

¹⁰ García-Valls & Lu

¹¹ Service level agreement

¹² Subramanian & Patabhi

۱-۵- تعریف مساله

امروزه با چالش‌های متنوعی در زمینه سیستم‌های رایانش ابری مواجه هستیم که یکی از این چالش‌ها چگونگی تخصیص منابع به منظور بهبود کیفیت سرویس و کاهش مصرف انرژی در مراکز داده ابری می‌باشد. افزایش مصرف انرژی در سیستم‌های رایانش ابری اثرات مخربی از جمله افزایش گرمای جهانی، آلودگی محیط و ... را در پی خواهد داشت. برای بیان مسئله خود در این تحقیق ما به دنبال راهکاری هستیم تا از سرریز شدن میزبان‌های فیزیکی جلوگیری کنیم. به این دلیل که سرریز شدن میزبان‌های فیزیکی نقض کیفیت سرویس را به همراه دارد. در این رساله قصد داریم به این مساله پردازیم که تخصیص ماشین‌های مجازی به میزبان‌های فیزیکی را به چه نحوی انجام دهیم که تا جای ممکن از سرریز شدن میزبان‌های فیزیکی با تخصیص مناسب ماشین‌های مجازی روی آن‌ها جلوگیری کنیم. همچنین به منظور مهاجرت ماشین‌های مجازی کنترلی روی آن‌ها به منظور مدیریت موثرتر صورت داده‌ایم.

۱-۶- اهداف تحقیق به صورت کلی و جزئی

هدف ما در این پایان نامه ارائه روشی برای کاهش میزبان‌های فیزیکی سرریز شده به منظور جلوگیری از نقض کیفیت خدمات و کاهش توان مصرفی می‌باشد. برای این منظور قصد داریم با جایابی بهینه ماشین‌های مجازی تا جای ممکن از سرریز شدن میزبان‌های فیزیکی جلوگیری کنیم. همچنین قصد داریم با کنترل مهاجرت، در بهبود مصرف انرژی و کیفیت سرویس تاثیر بگذاریم.

۱-۷- فرضیه‌های تحقیق

- در محیط مورد نظر فرض کردیم ماشین‌های مجازی و میزبان‌های فیزیکی از یک نوع نیستند. یعنی محیط ناهمگن است.
- درخواست‌ها هیچ وابستگی به هم ندارند و مستقل هستند
- هر درخواست روی یک ماشین مجازی قرار می‌گیرد.

۱-۸- ابزارهای اندازه‌گیری

برای ارزیابی روش پیشنهادی خود آن را با شبیه ساز کلاسیک مورد بررسی و ارزیابی قرار داده ایم .

۱-۹- جنبه نوآوری و جدید بودن تحقیق در چیست

در واقع قصد داریم با قرار دادن مناسب ماشین‌های مجازی به میزبانی که منابع لازم را برای آن ماشین مجازی دارد از اضافه باری آن میزبان جلوگیری کنیم. همچنین اگر میزبانی در آینده دچار اضافه باری شد با اعمال سیاستی مناسب برای انتخاب ماشین مجازی از آن میزبان بتوانیم در بهبود کیفیت سرویس موثر تر واقع شویم.

۱-۱۰- مراحل پایان‌نامه

در ادامه تحقیق، در فصل دوم به بررسی روش‌های قبلی بیان شده در زمینه کیفیت سرویس و مصرف انرژی می‌پردازیم. در فصل سوم، روش پیشنهادی به طور کامل شرح داده می‌شود. سپس در فصل چهارم به بررسی و ارزیابی روش پیشنهادی و کار مورد مقایسه می‌پردازیم. در نهایت، در فصل پنجم به جمع‌بندی پایان نامه و کارهای آینده می‌پردازیم.

فصل دوم: مروری بر روش‌های انجام شده

۲-۱- مروری بر روش‌های انجام شده

در مراکز داده ابری منابع مورد نیاز ماشین‌های مجازی ممکن است از ظرفیت سروری که روی آن‌ها میزبانی می‌شوند بیشتر شود. در نتیجه در مقیاس بزرگ این منابع نیاز به مدیریت خودکار دارند. انرژی مصرفی در محیط‌های ابری از دو جنبه مورد بررسی قرار می‌گیرد جنبه اول مدیریت استاتیک انرژی که بیشتر مربوط به تجهیزات و سخت‌افزاری می‌باشد. جنبه دوم مدیریت پویای مصرف انرژی می‌باشد. در محیط ابری عمل ترکیب پویای ماشین‌های مجازی با استفاده از مهاجرت ماشین‌های مجازی و خاموش کردن میزبان‌های فیزیکی بیکار باعث بهینه‌شدن مصرف منابع و کاهش مصرف انرژی می‌شود. با توجه به افزایش روزافزون محبوبیت سیستم‌های ابری، اگر انرژی ای که در منابع ارائه دهنده خدمات آن مصرف می‌شود کنترل نگردد، آنگاه هزینه ارائه سرویس‌های آن‌ها افزایش می‌یابد و در پی آن روی هزینه پرداختی سرویس گیرندگان تأثیر خواهد گذاشت. مسئله مهمتر اینکه این مسئله سهم زیادی در افزایش آلودگی محیط زیست خواهد داشت. لذا کشف راهکارهای بهره‌وری انرژی بسیار حیاتی است. در این فصل قصد داریم به بررسی روش‌های انجام شده در زمینه مدیریت ماشین‌های مجازی، بهبود کیفیت سرویس و کاهش مصرف انرژی بپردازیم.

۲-۲- روش چوی

در این مقاله (چوی و لی^{۱۳} ۲۰۱۶) یک مرکز داده که در آن ارائه‌دهنده خدمات، ماشین‌های مجازی را روی میزبان‌های فیزیکی برای مشترکان خود برای محاسبات در شکل تقاضا است، تامین می‌کنند. برای مرکز داده ابری، یک الگوریتم ترکیب کار مبتنی بر دسته بندی کار (به عنوان مثال محاسباتی و داده‌ای) و استفاده منابع (مثل CPU و RAM) پیشنهاد شده است. علاوه بر این، یک الگوریتم ترکیب ماشین مجازی برای تعادل زمان اجرای کار و مصرف انرژی بدون نقض توافق نامه سطح خدمات^{۱۴} (SLA) طراحی شده است. برخلاف تحقیقات موجود بر روی ترکیب ماشین‌های مجازی یا زمانبندی که از طرح‌های آستانه تک استفاده می‌کنند، در این مقاله بر روی طرح دو آستانه (بالا و پایین) که برای ترکیب ماشین مجازی استفاده می‌شود، تمرکز شده است. به طور خاص، زمانی که یک میزبان با استفاده از منابع کمتر از آستانه پایین عمل می‌کند، همه ماشین‌های مجازی روی میزبان برای مهاجرت به میزبان‌های دیگر زمانبندی خواهند شد و پس از آن میزبان مربوطه خاموش خواهد شد، در حالیکه زمانی که یک میزبان با بهره‌وری منابع بالاتر از حد بالای آستانه عمل می‌کند، یک ماشین مجازی برای جلوگیری از ۱۰۰ درصد استفاده از منابع مهاجرت داده

¹³ Choi & Lee

¹⁴ service level agreement

خواهد شد. براساس ارزیابی تجربی با داده‌های واقعی، ثابت شده که دسته بندی کارها براساس الگوریتم ترکیب انرژی محور به کاهش قابل توجه انرژی بدون نقض SLA دست یافته است.

۲-۳- روش چین

مهاجرت ماشین‌های مجازی در محیط محاسبات ابری یک موضوع مهم برای حل خیلی از مسائل مانند توازن بار است که می‌تواند با مهاجرت ماشین‌های مجازی از سرورهای بیش از حد بار شده و پربار و ترکیب سرورها که بار آنها بعد از مهاجرت به دیگر سرورها می‌تواند پایین آید. در این مقاله (چین^{۱۵} و دیگران ۲۰۱۶، ۶۲:۷۱) یک الگوریتم مهاجرت ماشین مجازی مبتنی بر حداقل سازی مهاجرت در رایانش ابری برای بهبود بهره‌وری و پاسخ نیازها برای کاربر و محدودیت در نقض سطح کیفیت سرویس که به فرم SLA شناخته می‌شود، پیشنهاد شده است. نتایج آزمایشات موثر بودن الگوریتم پیشنهاد شده را در مقایسه با الگوریتم‌های موجود نشان می‌دهد. اثر بخشی این تکنیک‌ها به حل خیلی از مسائل مثل موازنه بار، حفظ سیستم و غیره به منظور افزایش کارایی با استفاده از سیستم‌های ابری و همچنین کیفیت خدمات به مشتریان کمک می‌کند. در این مقاله یک الگوریتم تصمیم‌گیری کارآمد مهاجرت ماشین مجازی در محیط ابری برای حل مسائل بالا ارائه شده است.

۲-۴- روش باسکار

با رشد اخیر رایانش ابری، چالش بزرگ ارائه دهندگان سرویس مساله طراحی استراتژی موثری برای مدیریت منابع اشتراکی با برنامه‌های متفاوت است. مکانیزم مدیریت منابع باید اشتراک گذاری موثری از منابع را برای ماشین‌های مجازی با تضمین بهره برداری بهینه از منابع میزبان‌های فیزیکی در دسترس انجام دهد. مکانیزم مدیریت منابع به کاربران ابر و همچنین ارائه دهندگان خدمات اجازه می‌دهد که استفاده موثری از منابع در دسترس خود داشته باشند. این مقاله (باسکار^{۱۶} ۲۰۱۶، ۴۷۲) برنامه‌ای از مدل مجموعه راف برای فراهم کردن ماشین‌های مجازی پیشنهاد داده است. روش پیشنهاد شده از مشخصات/ دانش براساس روش‌های کاهش استفاده می‌کند. این روش قوانین را برای کاهش ویژگی‌های غیرضروری برای ماشین‌های مجازی تولید می‌کند. این قوانین به مدیریت ماشین‌های مجازی برای انتخاب موثر ماشین مجازی کمک می‌کند. این مقاله مشکلات تامین ماشین مجازی مورد تقاضا را مورد بررسی قرار داده است. تکنیک کاهش مبتنی بر

¹⁵ Chein

¹⁶ Bhaskar

دانش برای مساله تامین ماشین مجازی براساس منابع موجود را در نظر گرفته است. روش پیشنهاد شده قوانینی برای تصمیمات موثر در انتخاب و نگاشت برنامه‌ها به ماشین‌های مجازی برای مدیریت ماشین‌های مجازی تولید می‌کند.

۲-۵- گودرزی

در این کار (گودرزی و پدرام^{۱۷} ۲۰۱۶)، یک توافق‌نامه سطح خدمات (SLA) مبتنی بر روش مدیریت منابع برای مراکز داده ابری ارائه شده است، که انرژی سرورهای موجود، محدودیت اوج انرژی و مصرف توان خنک کننده‌ها را در نظر گرفته است. هدف این مدیر منابع به حداقل رساندن هزینه‌های عملیاتی مراکز داده است. ساختار سلسله مراتبی روش پیشنهاد شده مدیریت منابع را مقیاس پذیر می‌سازد. روش مدیریت منابع پیشنهاد شده به طور همزمان سرور و مصرف توان خنک کننده‌ها را در نظر می‌گیرد و پیچیدگی تصمیم گیری در مدیریت منابع و SLA را در سیستم‌های رایانش ابری تضمین می‌کند. در نظر گرفتن SLA و حالت مراکز داده در شناسایی مقدار منابع مورد نیاز برای تخصیص به برنامه‌ها باعث کاهش قابل توجهی در هزینه‌های عملیاتی مراکز داده شده است. اثربخشی طرح مدیریت پیشنهاد شده در مقایسه با کارهای قبلی با استفاده از یک ابزار شبیه سازی جامع نشان داده شده است. الگوریتم‌های مدیریت منابع پیشنهاد شده هزینه‌های عملیاتی مراکز داده را حدود ۴۰ درصد کاهش داده در حالی که SLA حفظ شده است و همچنین کاهش زمان اجرای الگوریتم‌های مدیریت تا ۸۶ درصد با توجه به روش مدیریت متمرکز را بیان می‌کند. در این مقاله یک ساختار سلسله مراتبی مدیریت منابع برای سیستم ابری پیشنهاد شده است. ساختار ارائه شده مقیاس پذیری و کارایی بالایی را در مقایسه با یک ساختار متمرکز در کارهای قبلی نشان می‌دهد. علاوه بر انعطاف پذیری مبتنی بر SLA با توجه به ویژگی ماشین‌های مجازی برای مساله مدیریت منابع، که یک فاکتور مهم برای عملکرد بالاتر روش در مقایسه با روش‌های قبلی است. علاوه بر این، از دست دادن کارایی روش غیرمتمرکز با توجه به نسخه متمرکز شده الگوریتم کمتر از ۲ درصد ۲۷ بار زمان اجرای کوتاهتری داشته است. نتایج الگوریتم پیشنهاد شده در تناسب انرژی بالاتر در کل مراکز داده، نقض SLA و هزینه مهاجرت کمتر و بهره وری سیستم‌های خنک کننده بالاتری را نتیجه شده است. ساختار مدیریت پیشنهاد شده برای مهاجرت ماشین‌های مجازی محلی و تنظیم تخصیص منابع برای جلوگیری از افزایش دما، اوج توان و شرایط SLA ضروری است.

¹⁷ Goudarzi & Pedram

۲-۶- روش اسماعیل

به منظور اجرای بهینه ترکیب ماشین‌های مجازی تحت محدودیت‌های کیفیت سرویس (QoS) مبتنی بر مصرف انرژی در مراکز داده ابری که حاوی منابع فیزیکی ناهمگن است، باید یک چارچوب که ترکیبی از بسیاری از الگوریتم‌های زیر سیستمی می‌باشد که شامل پیش‌بینی انتخاب، قرار دادن، و غیره است ایجاد شود. چندین استراتژی به منظور حداقل رساندن مصرف انرژی در محیط ابری می‌تواند استفاده شود، اما مهمتر از آن این است که به حداقل رساندن از طریق خاموش کردن میزبان انتخاب شده کم بار بعد از جابجایی همه ماشین‌های مجازی روی سرور انتخاب شده انجام می‌شود. پیش‌بینی منابع مورد نیاز در یک دوره زمانی معین در حال حاضر اولین و مهمترین گام در تامین پویا برای برآورد انتظارات QoS در بارکاری‌های متغیر می‌باشد. به عبارت دیگر، در این مقاله (اسماعیل و الخزرجی^{۱۸}، ۲۰۱۶، ۱:۶) از الگوهای استفاده شده قبلی برای برآورد بارکاری درخواست شده برای آینده ماشین مجازی در مراکز داده استفاده شده است. اولین گام در فرایند پیش‌بینی چارچوب مصرف انرژی به دسته داده‌های تاریخی (مهم) است. در این مقاله، یک دسته برای هر دو کاربر و درخواست‌های ماشین مجازی پیشنهاد شده است. بررسی گوگل واقعی که از ویژگی‌های بیش از ۲۵ میلیون کار جمع آوری شده بیش از یک دوره ۲۹ روزه به عنوان مثال در این مقاله استفاده شده است. نظارت باید برای جمع آوری داده از سطوح متفاوت از زیرساخت کل محاسبات (مثل ماشین مجازی، شبکه و ذخیره سازی) و منابع نرم افزاری (مثل وب سرور، دیتابیس سرور و برنامه ماشین مجازی) با استفاده از ابزاری مثل اپن‌استک استفاده شود. انرژی مصرف شده با هر بخش از سخت افزار در مراکز داده می‌تواند با استفاده از ابزاری مثل مدیر زیرساخت مراکز داده (DCIM) نظارت شود. روش ارائه شده در این مقاله برای پیش‌بینی ماشین مجازی دسته کاربر و دسته ماشین مجازی برای دست یافتن به پیش‌بینی بهتر مصرف انرژی مراکز داده ابری ترکیب شده است. الگوریتم فازی c-means نتایج بهتری را از روش مبتنی بر k-means برای هر دو دسته، دسته کاربر و ماشین مجازی برای تعداد کمی از دسته‌ها که بسیار مهم در کاهش تعداد ورودی در یک سیستم پیش‌بینی هستند نشان می‌دهد. صرف نظر از الگوریتم دسته بندی استفاده شده، دو هدف باید در نظر گرفته شود: کاهش خطا و حفظ سربار کم. به عبارت دیگر، اگرچه افزایش تعداد دسته‌ها در یک الگوریتم خطا را کاهش می‌دهد، این کار مساله پیش‌بینی و در نتیجه بهینه سازی مصرف انرژی را در مراکز داده ابری پیچیده می‌کند.

¹⁸ Ismaeel & Al-Khazraji

۷-۲- روش راجو

محاسبات ابری یک الگوی رایانشی توزیع شده در مقیاس بزرگ است که در آن یک استخراج منابع به صورت پویا مقیاس پذیر و مجازی مثل توان محاسباتی، ذخیره سازی، سیستم عامل و سرویس و تقاضا برای مشتریان خارجی از طریق اینترنت تحویل داده می شود. در زمانبندی محاسبات ابری فرایند تصمیم گیری برای تخصیص منابع در قالب ماشین های مجازی برای برنامه های درخواست شده می باشد. در این مقاله (راجو و موسی^{۱۹}، ۲۰۱۶، ۴) دو مرحله زمانبندی مهلت آگاه برای زمانبندی ماشین های مجازی برای برنامه های درخواست شده در محاسبات ابری از مشتریان دریافت شده پیشنهاد شده است. در این مدل هر برنامه به دو نوع ماشین مجازی برای تکمیل آن کار نیاز دارد. این مدل ماشین های مجازی را به عنوان منابع برای برنامه (جاب) های درخواست شده مبتنی بر زمان پردازش و زمانبندی برنامه ها با در نظر گرفتن مهلت با توجه به زمان پاسخ و زمان انتظار تخصیص می دهد. یک محیط شبیه سازی توسعه داده شده و ارزیابی شده برای ارزیابی این مدل با در نظر گرفتن معیارهای ارزیابی از میانگین زمان چرخش، میانگین زمان انتظار و نقض در مهلت زمانی که با الگوریتم های اول بهترین (FCFS) و استراتژی زمانبندی کوتاهترین اول (SJF) مقایسه شده است. این مدل معیارهای ارزیابی را با فاکتور ثابت در مقایسه با سایر روش های زمانبندی کاهش می دهد. زمانبندی n جاب روی دو نوع از ماشین های مجازی با استفاده از الگوریتم زمانبندی مهلت آگاه دو مرحله ای عملکرد بهتری را در مقایسه با دیگر روش های زمانبندی می دهد. نتایج تجربی نشان می دهد که الگوریتم زمانبندی دو مرحله ای مهلت آگاه زمان انتظار میانگین، زمان برگشت میانگین، نقض مهلت میانگین با توجه به زمان انتظار، میانگین نقض مهلت با توجه به زمان پاسخ به طور معقولی در مقایسه با روش های FCFS و SJF و الگوریتم های زمانبندی دو مرحله ای کاهش می دهد. تعداد نقض مهلت جاب ها با توجه به زمان پاسخ و زمان انتظار با در نظر گرفتن فاکتور ثابت در الگوریتم دو مرحله ای مهلت آگاه در مقایسه با الگوریتم های قبلی کاهش یافته است.

۸-۲- روش دان

یکی از چالش های موجود در زمینه سیستم های ابری، چگونگی کاهش مصرف انرژی با حفظ ظرفیت محاسباتی بالا است. روش های موجود اساساً بر روی افزایش بهره برداری منابع تمرکز کرده اند. برنامه های کاربردی با منابع مورد نیاز متفاوتی بر روی ماشین های مجازی اجرا می شوند که بر روی کارایی سیستم

¹⁹ Raju & Moses

و مصرف انرژی تأثیر می‌گذارند. همچنین ممکن است که اوج بار ^{۲۰} لحظه‌ای منجر به این شود که در سودمندی مصرف انرژی تأثیر بگذارد. در تحقیق دیگری (دان و وو ^{۲۱} ۲۰۱۶) الگوریتم زمانبندی جدیدی با نام PreAntPolicy ارائه شده است که شامل مدل پیش‌بینی براساس مکانیزم‌های فرکتال ^{۲۲} و زمانبندی براساس بهبود الگوریتم کلونی است. محققین مقاله با استفاده از تحلیل‌های زیاد و آزمایشات شبیه‌سازی در بارکاری واقعی محاسبات کلاسترهای گوگل توانستند کارایی کار خود را در سودمندی مصرف انرژی و بهره‌وری منابع نشان دهند. علاوه بر این روش پیشنهادی محققین مقاله مدل ذخیره تأمین ظرفیت پویای مؤثری را برای برنامه‌های کاربردی با نیازهای منابع متفاوت در محیط محاسبات ناهمگن را پیشنهاد می‌کند که می‌تواند مصرف منابع سیستم و انرژی را کاهش دهد به‌طوری‌که زمانبندی مناسبی را در زمان اوج بار فراهم می‌کند. در آزمایشات شبیه‌سازی خود از الگوریتم‌های زمانبندی اول بهترین حریصانه ^{۲۳}، نوبت چرخشی ^{۲۴} (که معمولاً توسط برخی از محاسبات ابری استفاده می‌شود) و حداقل توان مهاجرت استفاده کردند. نتایج شبیه‌سازی نشان می‌دهد که روش پیشنهادی مقاله در مقایسه با الگوریتم اول بهترین ۱۷/۷۶٪ و در مقایسه با الگوریتم نوبت چرخشی ۱۸/۷۵٪ کاهش در مصرف انرژی داشته‌است، در حالیکه از نقض کیفیت سرویس درخواست شده تا جای ممکن جلوگیری شده است.

۲-۹- روش پاتل

یکی از چالش‌های مهم در سیستم‌های ابری، تخصیص منابع است. در تحقیق دیگری (پاتل ^{۲۵} ۲۰۱۵) الگوریتمی به نام بهترین کاهش اصلاح شده ^{۲۶} به صورت الگوریتم انرژی محور EABFD پیشنهاد شده است. روش EABFD در ابتدا دو صف از میزبان‌های فیزیکی کم بار و خالی را تشکیل می‌دهد. صف میزبان‌های فیزیکی خالی و صف میزبان‌های کم بار را در ابتدا با هدف بهبود تخصیص ماشین‌های مجازی مقداردی اولیه می‌کند. طبق این الگوریتم، همه ماشین‌های مجازی براساس کاهش بهره‌وری از پردازنده آنها مرتب می‌شوند. سپس این الگوریتم، بهترین میزبان فیزیکی را در میان همه میزبان‌های کم بار و خالی پیدا می‌کند. برای این منظور، در ابتدا، میزبان‌های کم بار را بررسی می‌کند، در نهایت، اگر در میان همه میزبان‌های کم بار، میزبان فیزیکی مورد نیاز را پیدا نکند، این الگوریتم یک میزبان از میزبان‌های خالی لیست

²⁰ Peak loads

²¹ Duan & Wu

²² Fractal

²³ Greedy First-Fit (FF)

²⁴ Round-Robin (RR)

²⁵ Patel

²⁶ Modified best fit decreasing

برای تخصیص ماشین مجازی روی آن را روشن می‌کند. این الگوریتم تلاش دارد تعداد میزبان‌های روشن را به منظور کاهش مصرف انرژی حداقل کند. در این مقاله صرفه جویی در میزان انرژی با ترکیب موثر ماشین‌های مجازی انجام می‌شود. در جدول (۱-۲) مقالات بیان شده براساس سه معیار خلاصه شده است.

جدول ۱-۲ مقایسه مقالات بر اساس سه معیار

مرجع	معیار کاهش انرژی	معیار بهبود کیفیت سرویس	معیار کاهش تعداد مهاجرت‌ها
[۴]	✓	✓	
[۳]			✓
[۲]	✓		✓
[۸]		✓	✓
[۱۰]	✓		
[۱۳]		✓	
[۶]	✓		
[۱۲]	✓		✓

فصل سوم: روش پیشنهادی

۳-۱- مقدمه

بهبود و حفظ کیفیت سرویس یکی از موضوعات مهم در زمینه سیستم‌های رایانش ابری است. برای این منظور، نیاز است تا برنامه‌ریزی‌های مختلف و سیاست‌های متفاوتی در زمینه مدیریت این سیستم‌ها در نظر گرفته شود تا بتوانیم با مدیریت مناسب منابع از افزایش مصرف انرژی و نقض شدن کیفیت سرویس جلوگیری کنیم. اگر مدیریت مناسب و روش‌های مناسبی در جای‌دهی ماشین‌های مجازی به ماشین‌های فیزیکی صورت گیرد می‌توانیم در بهبود کیفیت سرویس و انرژی مصرفی تاثیر بگذاریم. سوالاتی که قصد داریم در این تحقیق به آن‌ها پردازیم به شرح زیر است:

- به چه نحوی می‌توانیم جای‌دهی مناسبی از ماشین‌های مجازی روی ماشین‌های فیزیکی فراهم آوریم؟
 - به چه نحوی در استفاده مناسب از منابع ماشین‌های فیزیکی تاثیر می‌گذاریم؟
 - چه روشی برای کنترل مهاجرت ماشین‌های مجازی به منظور انتخاب ماشین مجازی مناسب استفاده کنیم؟
- آنچه در این پایان‌نامه قصد داریم به آن توجه کنیم شامل جای‌دهی مناسب ماشین‌های مجازی به ماشین‌های فیزیکی و مدیریت کردن مهاجرت ماشین‌های مجازی می‌باشد. در ادامه به بررسی روش پیشنهاد شده و پارامترهای مورد ارزیابی می‌پردازیم.

۳-۲- تجزیه و تحلیل روش پیشنهادی

سیستم رایانش ابری مورد استفاده در روش پیشنهادی یک محیط سطح IaaS با ماشین‌های فیزیکی متنوع و ناهمگن می‌باشد. در سیستم‌های ابری، چندین کاربر مستقل درخواست‌هایشان را برای N ماشین مجازی ناهمگن که توان پردازشی آنها (بهره پردازنده) در واحد $^{27}MIPS$ تعریف می‌شود و همچنین مقدار حافظه و پهنای باند شبکه است، ارسال می‌کنند. به این ترتیب فراهم آورنده‌ی ابر باید بر روی میزان منابع داده شده به ماشین مجازی و بار آن و نیز تغییرات مصرف انرژی ماشین فیزیکی مورد نظر نظارت داشته باشد. برای بیان روش خود، درخواست‌ها بر روی ماشین‌های مجازی قرار می‌گیرند. ماشین‌های مجازی به صورت مجموعه $VM = (VM_1, VM_2, \dots, VM_n)$ در نظر گرفته می‌شوند و این ماشین‌های مجازی بر روی m

²⁷Millions Instructions Per Second

ماشین فیزیکی به صورت $PM = (PM_1, PM_2, \dots, PM_n)$ قرار می‌گیرند. زمانی که ماشین‌های مجازی بر روی ماشین‌های فیزیکی قرار می‌گیرند، بعد از این جای دهی ممکن است یک میزبان با استفاده زیاد از منابع آن دچار اضافه باری شود و نتواند به درخواست کاربر پاسخ دهد و نقض کیفیت سرویس را ایجاد می‌کند. در این کار سعی داریم با جاییابی بهینه ماشین‌های مجازی تا جای ممکن از سرریز شدن میزبان‌های فیزیکی جلوگیری کنیم.

هدف ما در این پایان نامه ارائه روشی برای کاهش ماشین‌های فیزیکی سرریز شده به منظور جلوگیری از نقض کیفیت خدمات و کاهش توان مصرفی می‌باشد. به منظور جای دهی مناسب ماشین‌های مجازی به ماشین‌های فیزیکی با توجه به منابع مورد نیاز ماشین مجازی و منابع در دسترس میزبان فیزیکی، میزان منابع اختصاص داده شده به ماشین مجازی را تخمین می‌زنیم و هر ماشین مجازی به میزبانی تخصیص می‌دهیم که منابع اختصاص داده شده به آن ماشین مجازی بیشتر از میزبان درخواست شده توسط آن ماشین مجازی باشد. برای این منظور معیاری که در فرمول (۱-۳) در زیر بیان شده است را مطرح می‌کنیم.

$$Factor = \frac{VM_{resource\ requirements}}{PM_{available\ resource}} \quad (1-3)$$

برای هر ماشین فیزیکی این معیار را محاسبه می‌کنیم و میزبانی را به عنوان میزبان مورد نظر برای جای دهی ماشین مجازی انتخاب می‌کنیم که کمترین مقدار را در بین دیگر ماشین‌های فیزیکی دارد. علت این انتخاب این است که هرچه مقدار این معیار کمتر باشد نشان دهنده این است که منابع موجود ماشین فیزیکی نسبت به منابع مورد نیاز ماشین مجازی بیشتر است و احتمال کمتری وجود دارد که آن میزبان دچار سرریزی شود.

۳-۳- روش ترکیب ماشین‌های مجازی

بعد از اینکه ماشین‌های مجازی به ماشین فیزیکی مناسب تخصیص داده می‌شود، مسئله‌ی ترکیب پویای ماشین‌های مجازی به منظور استفاده بهینه از منابع و بهبود کیفیت سرویس به ۳ بخش تقسیم می‌شود که شامل شناسایی میزبان‌های فیزیکی که به عنوان پربار در نظر گرفته می‌شوند که نیاز به مهاجرت یک یا چند ماشین مجازی از این میزبان فیزیکی برای جلوگیری از نقض کیفیت خدمات می‌باشد. در مرحله بعد انتخاب ماشین مجازی از ماشین فیزیکی سرریز شده می‌باشد تا از نقض کیفیت سرویس جلوگیری شود. گام بعدی شناسایی زمانی که یک میزبان فیزیکی به عنوان کم‌بار²⁸ در نظر گرفته می‌شود که نیاز به مهاجرت

²⁸Under load

همه‌ی ماشین‌های مجازی از این ماشین فیزیکی دارد و تغییر حالت ماشین فیزیکی به حالت خاموش است (چین ۲۰۱۶، ۶۲:۷۱ و پاتل ۲۰۱۵). همچنین ما قصد داریم کنترلی در سیاست انتخاب ماشین مجازی از ماشین فیزیکی برابر اعمال کنیم. برای این منظور، بعد از اینکه ماشین‌های مجازی به ماشین فیزیکی مناسب تخصیص داده می‌شود ممکن است باز هم میزبانی وجود داشته باشد که دچار اضافه باری شود. براساس مقاله (چین ۲۰۱۶، ۶۲:۷۱ و پاتل ۲۰۱۵) از حد آستانه بالا²⁹ برای شناسایی ماشین‌های سرریز شده استفاده می‌کنیم.

این حد آستانه به صورت پویا براساس بار قرار گرفته روی هر میزبان تعریف می‌شود. زمانی که یک میزبان دچار اضافه‌بار می‌شود نیاز است یک یا تعدادی از ماشین‌های مجازی آن میزبان به منظور جلوگیری از نقض کیفیت خدمات کاربر مهاجرت داده شود. در مقاله پاتل سه سیاست برای انتخاب ماشین مجازی از ماشین فیزیکی سرریز شده ارائه شده است. روش اول MU یا حداقل بهره پردازنده، که در این روش ماشین مجازی برای مهاجرت از میزبان سرریز شده انتخاب می‌شود که دارای حداقل استفاده از پردازنده است. روش دوم روش تصادفی است که یک ماشین مجازی به صورت تصادفی انتخاب می‌شود. روش بعدی برای مهاجرت ماشین‌های مجازی، روش زمان مهاجرت حداقل³⁰ (MMT) نام دارد. در این روش یک ماشین مجازی که مقدار حافظه به پهنای باند کمتری را دارد را برای مهاجرت انتخاب می‌کند. فرمول (۲-۳) این سیاست را بیان می‌کند (پاتل ۲۰۱۵):

$$v \in V_j | \forall a \in V_j, \frac{RAM_u(v)}{NET_j} \leq \frac{RAM_u(a)}{NET_j} \quad (2-3)$$

زمان مهاجرت با مقدار RAM استفاده شده VM تقسیم بر پهنای باند شبکه در دسترس برای ماشین فیزیکی زبرآورد می‌شود. یک مجموعه از VM های است که اخیراً به میزبان فیزیکی اختصاص یافته است. مقدار RAM استفاده شده اخیر توسط VM_a است. پهنای باند شبکه در دسترس برای میزبان فیزیکی Z است. ما سعی داریم تغییری در این سیاست‌ها اعمال کنیم تا بتوانیم در انتخاب ماشین مجازی مناسب موثرتر واقع شویم. اگر چندین ماشین مجازی مقدار حافظه یکسان داشته باشند در روش MMT

فاکتوری را برای این حالت در نظر نگرفته است. ما قصد داریم زمانی که این حالت اتفاق افتاد ماشین مجازی که استفاده از پردازنده بیشتری دارد را برای مهاجرت انتخاب کنیم. زیرا با این انتخاب آن میزبان فیزیکی احتمال بیشتری دارد که از حالت اضافه باری خارج شود. همچنین در حالت MU اگر چند ماشین مجازی دارای بهره پردازنده یکسان بودند آن ماشین مجازی را انتخاب کنیم که حداقل مقدار حافظه را

²⁹Upper threshold

³⁰Minimum Migration Time

دارد تا زمان مهاجرت را حداقل کرده و از نقض کیفیت سرویس جلوگیری کنیم. برای شناسایی میزبان‌های فیزیکی با بار کمتر از حد نرمال، طبق روش پاتل ماشین فیزیکی که نسبت به دیگر ماشین‌های فیزیکی از منابع خود کمتر استفاده می‌کند به عنوان کم بار در نظر گرفته می‌شود. در نهایت برای قرار دادن ماشین‌های مجازی از این ماشین فیزیکی روی ماشین‌های فیزیکی دیگر تلاش می‌کند و ماشین فیزیکی مبدأ زمانی که همه‌ی ماشین‌های مجازی مهاجرت داده شد به حالت خواب تغییر پیدا می‌کند.

۳-۴- بدست آوردن میزان انرژی مصرفی

برای بدست آوردن میزان انرژی استفاده شده توسط ماشین‌های فیزیکی از فرمول ارائه شده در روش پاتل استفاده می‌کنیم. طبق آزمایشات انجام شده، بهره‌وری و استفاده از پردازنده در مقایسه با دیگر منابع یک ماشین فیزیکی انرژی مصرف می‌کند. برای این منظور فرمولی که برای محاسبه انرژی مصرفی ماشین فیزیکی بیان شده است براساس بهره‌وری و استفاده از پردازنده می‌باشد. فرمول (۳-۳) در رابطه زیر، فرمول انرژی را بیان می‌کند: (پاتل ۲۰۱۵)

$$E = \int_{t_0}^{t_1} P(u(t)) dt \quad (3-3)$$

طبق فرمول بالا، از آنجا که استفاده از پردازنده ممکن است با گذشت زمان به علت تغییرپذیری بار کاری، تغییر کند، از این‌رو، بهره‌وری پردازنده تابعی از زمان است و به عنوان $u(t)$ ارائه می‌شود. E به صورت انتگرال تابع مصرف انرژی روی یک دوره زمانی تعریف می‌شود که در رابطه بالا نمایش داده شده است. در روش پیشنهادی طبق فرمول بالا به محاسبه مصرف توان ماشین‌های فیزیکی و سپس مصرف انرژی آنها به صورت منفرد محاسبه شده و به صورت زیر مجموع مصرف انرژی ابر را محاسبه می‌کنیم: (پاتل ۲۰۱۵)

$$ET_t = \sum_{i=1}^n E_i \quad (4-3)$$

طبق فرمول بالا، n تعداد کل ماشین‌های فیزیکی، E_i انرژی مصرف شده توسط میزبان i ام تا زمان t ، ET_t مجموع کل انرژی مصرفی ابر در زمان t است.

۳-۵- بدست آوردن میزان نقض کیفیت سرویس

کیفیت سرویس بحث مهمی در زمینه سیستم‌های ابری است. نقض شدن کیفیت درخواست شده از طرف کاربر برای فراهم آورنده‌ی ابر بسیار نامطلوب خواهد بود. به این دلیل که باید در مقابل کیفیت سرویس نقض شده جریمه‌های مالی پرداخت شود. کیفیت سرویس در محیط ابر معمولاً به فرم SLA (توافق نامه سطح خدمات) شناخته می‌شوند. از آنجایی که بر روی یک ماشین فیزیکی بیش از ظرفیت آن ماشین مجازی قرار داده شده است، پارامتری که می‌تواند مورد نظارت قرارگیرد، میزان مصرف منابع آن ماشین فیزیکی می‌باشد. در محیط ابری عواملی مثل مهاجرت و سربار شدن میزبان‌های فیزیکی باعث نقض خدمات می‌شود. برای این منظور از دو پارامتر طبق فرمول ارائه شده در روش‌های پاتل و بلوگلازوف برای محاسبه نقض کیفیت خدمات استفاده می‌کنیم. این دو پارامتر شامل: زمان نقض SLA هر میزبان (SLATAH) زمانی که میزبان‌ها از تمام بهره خود استفاده می‌کنند که باعث نقض خدمات می‌شود. فرمول ۳-۵ برای این منظور در زیر بیان شده است: (بلوگلازوف ۲۰۱۳^{۳۱}، ۱۴۲۰: ۱۳۹۷)

$$SLATAH = \frac{1}{N} \sum_{i=1}^N \frac{T_{si}}{T_{ai}} \quad (5-3)$$

N تعداد ماشین‌های فیزیکی، T_{si} زمان کل در طولی که ماشین فیزیکی i از تمام بهره خود استفاده می‌کند که نقض کیفیت خدمات را ایجاد می‌کند. T_{ai} زمان کل ماشین فیزیکی i که در حالت فعال است. پارامتر بعدی کاهش کارایی کل با مهاجرت ماشین‌های مجازی (PDM) است که مربوط به زمانی است که مهاجرتی صورت می‌گیرد که باعث نقض کارایی می‌شود: (تانی و عمرانی ۲۰۱۶^{۳۲}، ۱۸۶۶)

$$PDM = \frac{1}{M} \sum_{j=1}^m \frac{C_{dj}}{C_{rj}} \quad (6-3)$$

M تعداد VM ها، C_{dj} نقض کارایی VM_j که با مهاجرت ایجاد می‌شود را برآورد می‌کند. C_{rj} کل ظرفیت پردازنده درخواست شده توسط VM_j در طول دوره زندگی آن است. در آزمایشات C_{dj} با ۱۰٪ از بهره پردازنده در MIPS در طول مهاجرت همه‌ی VM_j برآورد شده است.

این دو معیار هر دو در نقض کیفیت سرویس موثر هستند. هم زمانی که یک ماشین فیزیکی دچار اضافه

³¹ Beloglazov

³² Tani & El Amrani

باری می‌شود و هم زمانی که مهاجرتی صورت می‌گیرد. برای این منظور از یک معیار ترکیبی که شامل هر دو معیار است استفاده می‌شود (بلوگرلازو و ۲۰۱۵)

$$SLAV = SLATAH.PDM \quad (۷-۳)$$

در این فصل روش پیشنهادی به طور کامل شرح داده شد. در فصل بعدی پارامترهای ارزیابی تعریف میشود و روش پیشنهادی به کمک آنها مورد ارزیابی قرار می‌گیرد.

فصل چهارم: بررسی و ارزیابی راه حل پیشنهادی

۴ - ۱ - محیط آزمایش

به منظور بررسی و ارزیابی کار خود و روش مورد مقایسه، شبیه ساز انتخاب شده کلودسیم^{۳۳} ورژن ۳/۰ می باشد که یکی از ابزارهای مهم و معروف شبیه سازی در سیستم های ابری می باشد. کلودسیم یک چارچوب شبیه سازی جدید، عمومی و قابل توسعه می باشد. این ابزار به عنوان یک چارچوب شبیه سازی در دانشگاه Melbourne توسعه یافته است. امکان مدلسازی بدون لایه، شبیه سازی روی زیرساخت طراحی شده محاسبات ابری را فراهم می آورد. این ابزار پلتفرمی است که می تواند برای مدل کردن مراکز داده، ماشین های فیزیکی، ماشین های مجازی، سیاست های زمانبندی و تخصیص ماشین های مجازی به میزبان های فیزیکی استفاده شود. این چارچوب یک موتور مجازی سازی را با جنبه های افزوده ای برای مدلسازی ایجاد و مدیریت موتورهای مجازی در یک مرکز داده ای ارائه می کند (تانی و آل عمرانی ۲۰۱۶، ۴). به منظور شبیه سازی روش خود، محیط را ناهمگن در نظر گرفته ایم. برای این منظور، طبق مقاله پاتل که به عنوان مقاله پایه در نظر گرفته شده است، ماشین های فیزیکی را در دو حالت در نظر گرفته ایم. در حالت اول، بهره پردازنده با ۱۸۶۰ میلیون دستورالعمل در ثانیه (MIPS)^{۳۴} می باشد و در حالت دوم بهره پردازنده ماشین فیزیکی با ۲۶۶۰ میلیون دستورالعمل در ثانیه می باشد. مقدار حافظه RAM، ۴ گیگابایت و پهنای باند شبکه ۱ GB/s برای هر ماشین فیزیکی در نظر گرفته ایم. ماشین های مجازی نیز دارای ویژگی های ناهمگن می باشند. برای ماشین های مجازی نیز ظرفیت پردازشی ۵۰۰، ۱۰۰۰، ۲۰۰۰ و پهنای باند ۱۰۰۰۰ در نظر گرفته ایم. در جدول (۴ - ۲) و (۴ - ۳) مشخصات ماشین های مجازی و میزبان های فیزیکی در قالب جدول بیان شده است.

^{۳۳} CloudSim

^{۳۴} Million Instructions Per Second

جدول ۴ - ۲ مشخصات ماشین‌های مجازی

VM	Ram	MIPS	PesNumber	BW
۰	۸۷۰	۵۰۰	۱	۱۰/۰۰۰
۱	۱۷۴۰	۱/۰۰۰	۱	۱۰/۰۰۰
۲	۱۷۴۰	۲/۰۰۰	۱	۱۰/۰۰۰

جدول ۴ - ۳ مشخصات میزبان فیزیکی

Host	Ram	MIPS	PesNumber	BW
۰	۴GB	۲۶۶۰	۲	۱/۰۰۰/۰۰۰
۱	۴GB	۱۸۶۰	۲	۱/۰۰۰/۰۰۰

۴ - ۲ - نتایج مربوط به شبیه سازی

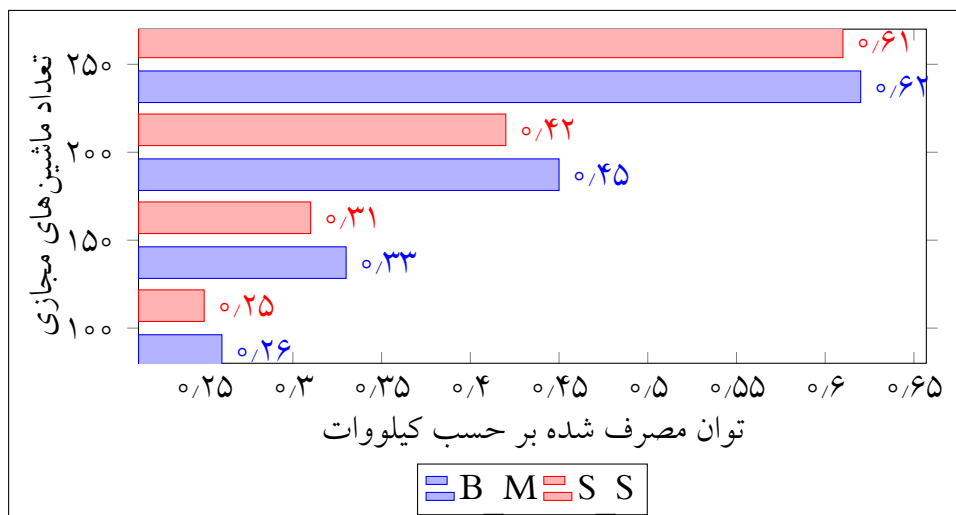
در نمودارهای مورد آزمایش، برای بیان کردن روش خود از واژه S_S^{۳۵} و برای بیان روش مورد مقایسه از واژه B_M^{۳۶} استفاده کرده ایم.

برای مقایسه کار خود و روش مورد مقایسه طبق شبیه ساز کلودسیم به بررسی انرژی مصرف شده در کل اجرای برنامه و نقض کیفیت سرویس رخ داده شده که در فصل ۲-۹ آن را بررسی کردیم، پرداخته ایم.

در نمودار (۴-۱) و (۴-۲) به بررسی توان مصرف شده با سیاست MMT و MU پرداخته ایم. توان مصرفی کل مراکز داده بر حسب کیلو وات اندازه گیری می شود. به منظور مقایسه کار خود، کار خود و مقاله پایه را با تعداد ماشین های مجازی متفاوتی که شامل ۱۰۰، ۱۵۰، ۲۰۰، ۲۵۰ می باشد مورد بررسی قرار داده ایم.

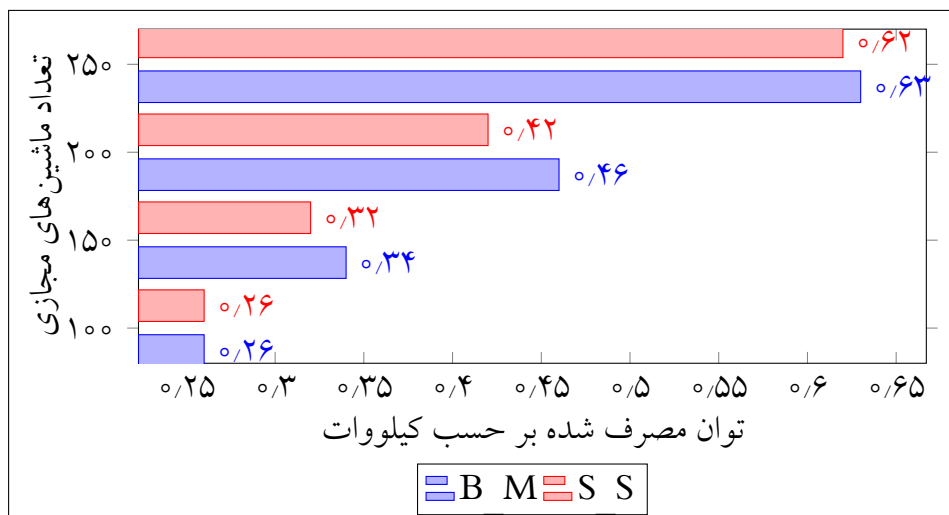
^{۳۵} Suggested solution

^{۳۶} Basic method



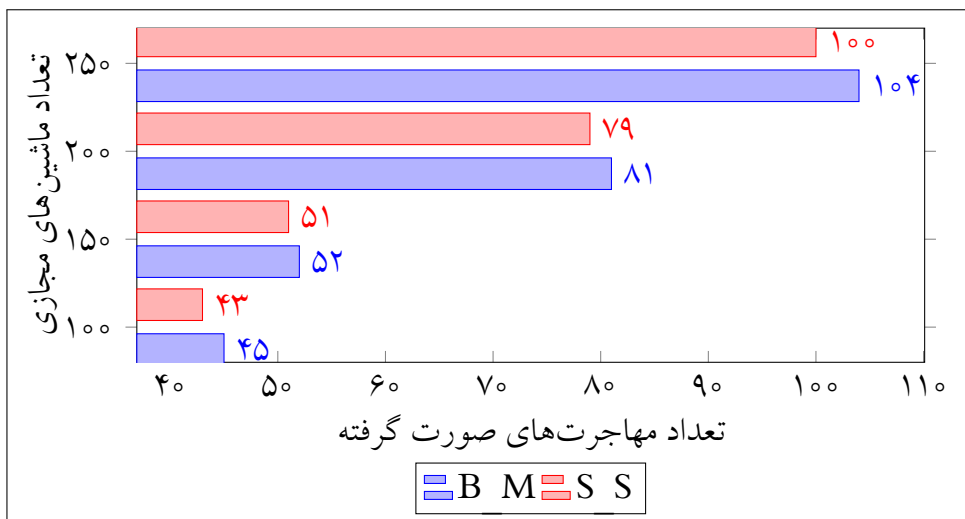
نمودار ۴-۱ مقایسه مصرف انرژی با سیاست MMT

همانطور که از نمودار (۴-۱) و (۴-۲) ملاحظه می شود، با تعداد متفاوتی از ماشین های مجازی در حالات مختلف توان مصرف شده روش پیشنهادی نسبت به کار مورد مقایسه کاهش داشته است. دلیل این کاهش در این است که ما در ابتدا زمانی که ماشین های مجازی را به میزبان های فیزیکی تخصیص دادیم سعی کردیم از منابع میزبان های فیزیکی مناسب استفاده کنیم. سعی کردیم با اعمال جای دهی مناسب در حفظ تعادل بار که در بهبود توان مصرف تاثیر گذار است، موثر واقع شویم.

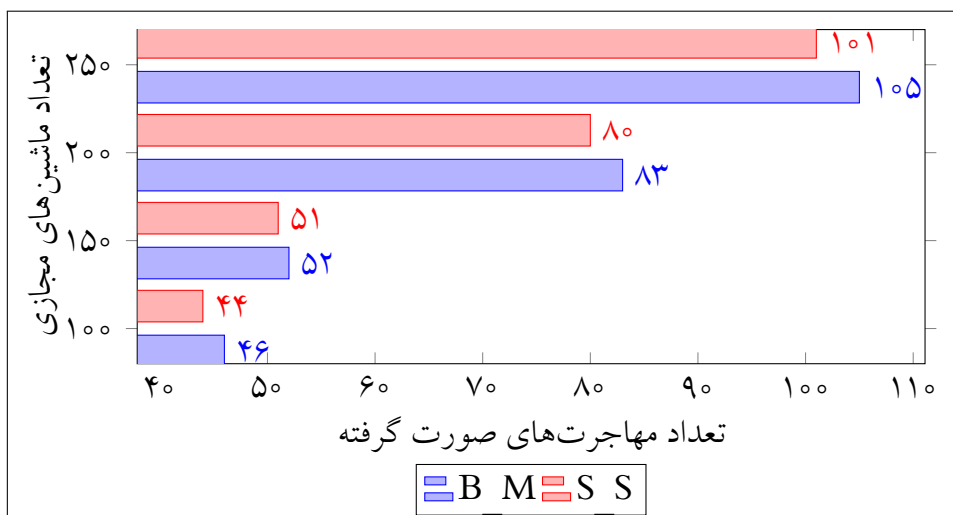


نمودار ۲-۴ مقایسه مصرف انرژی با سیاست MU

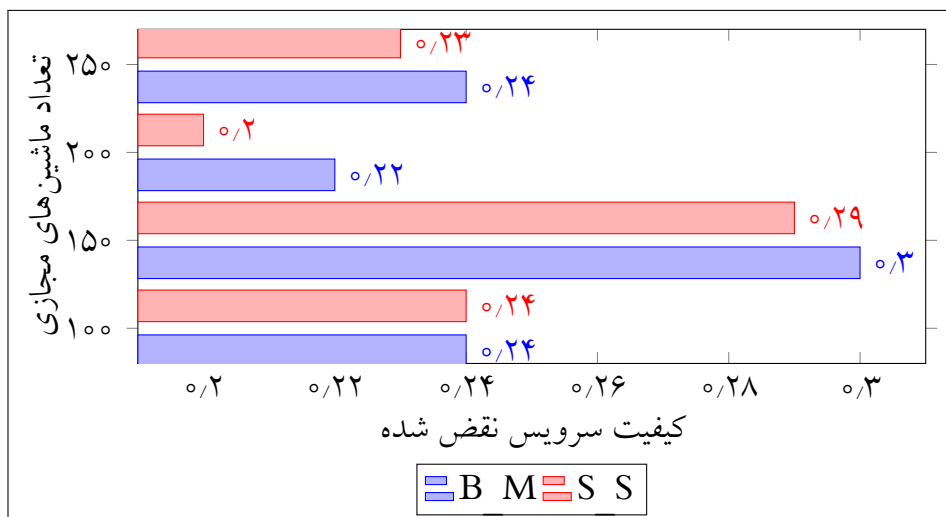
در نمودار (۳-۴) و (۴-۴) به بررسی تعداد مهاجرت های رخ داده در کل اجرای برنامه ها پرداخته ایم. تعداد ماشین های مجازی ۱۰۰، ۱۵۰، ۲۰۰ و ۲۵۰ در نظر گرفته شده است. همانطور که در نمودار (۳-۴) ملاحظه می شود، به ازای تعداد مختلف ماشین های مجازی روش پیشنهادی بهبودی در تعداد مهاجرت های رخ داده نسبت به روش پایه داشته است. علت این بهبود در این است که با تغییراتی در سیاست های MMT و MU و با انتخاب سیاست مناسب در انتخاب ماشین مجازی برای مهاجرت و جای دهی مناسب سعی کردیم در کاهش تعداد مهاجرت ها که عامل موثری در نقض کیفیت سرویس می باشد، تاثیر بگذاریم.



نمودار ۳-۴ تعداد مهاجرت‌های رخ داده با سیاست MMT

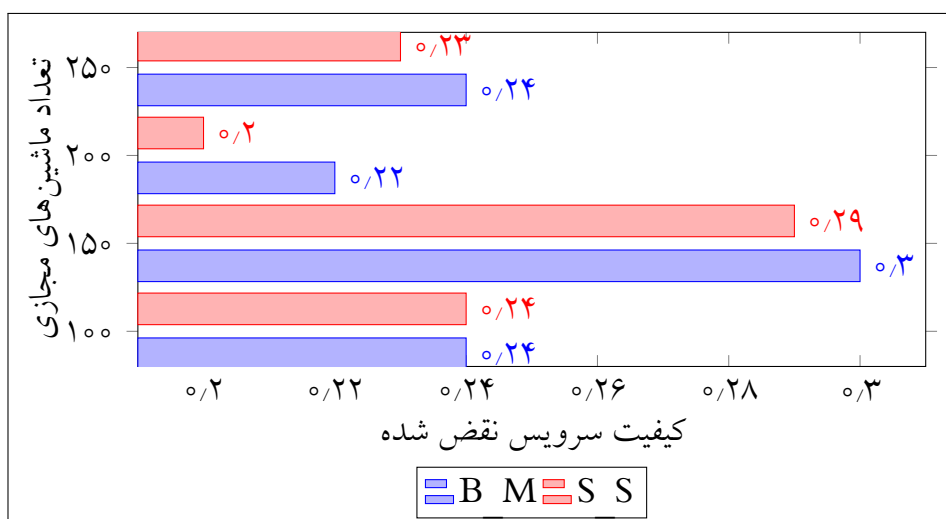


نمودار ۴-۴ تعداد مهاجرت‌های رخ داده با سیاست MU



نمودار ۴-۵ مقایسه کیفیت سرویس نقض شده در روش پیشنهادی و مورد مقایسه با سیاست MMT

در نمودار (۴-۵) و (۴-۶) به بررسی کیفیت سرویس نقض شده پرداخته ایم کیفیت سرویس معمولاً به فرم SLA در محیط ابری شناخته می شود. تعداد ماشین های مجازی ۱۰۰، ۱۵۰، ۲۰۰، ۲۵۰ در نظر گرفته شده است. همانطور که در نمودار (۴-۵) مشاهده می شود، روش پیشنهادی نقض کیفیت سرویس کمتری در مقایسه با روش مورد مقایسه دارد. در روش S_S با انتخاب مناسب ماشین مجازی برای مهاجرت و جای دهی مناسب ماشین های مجازی سعی کردیم از منابع ماشین های فیزیکی به طور موثر بهره مند شویم و احتمال وقوع نقض کیفیت سرویس را بهبود بخشیم. در این فصل به بررسی و شبیه سازی روش پیشنهادی و روش مورد مقایسه پرداختیم. در فصل بعد به نتیجه گیری و کارهای آینده می پردازیم.



نمودار ۴-۶ مقایسه کیفیت سرویس نقض شده در روش پیشنهادی و مورد مقایسه با سیاست MU

فصل پنجم: جمع بندی و کارهای آینده

۵-۱- جمع بندی و کارهای آینده

امروزه سیستم های پردازش ابری یکی از موضوعات حیاتی و مهم در زمینه فناوری اطلاعات می باشد. به کارگیری این تکنیک در کاهش هزینه ها، کاهش زمان اجرا و تاثیر گذار است. مباحثی مانند توان مصرفی مراکز داده، زمان پاسخ، کیفیت سرویس کاربر و هزینه ها از مباحث مهمی است که در حوزه سیستم های پردازش ابری مورد توجه زیادی قرار گرفته است. در نتیجه استفاده از راهکارهای موثر و مدیریت مناسب ماشین های مجازی و کنترل مهاجرت های رخ داده می تواند در کاهش مواردی مانند توان مصرفی، تعداد مهاجرت ها و نقض کیفیت سرویس تاثیر بگذارد. کارهای زیادی در حوزه بهبود بهره وری انرژی و کیفیت سرویس در مراکز داده ابری صورت گرفته است. روش هایی همچون جای دهی و ترکیب پویای ماشین های مجازی در مراکز داده ابری روش های موثری برای کاهش توان مصرفی می باشد. روش های مربوط به ترکیب پویای ماشین های مجازی این ویژگی را فراهم می کند تا با استفاده از امکان مهاجرت ماشین های مجازی از ماشین های فیزیکی حداقلی در مراکز داده استفاده شود. در این پایان نامه ما سعی کردیم با استفاده مناسب از منابع موجود ماشین های فیزیکی و جای دهی درست و مناسب و در نهایت با انتخاب ماشین مجازی مناسب به منظور مهاجرت به اهداف همچون بهبود توان مصرفی و کیفیت خدمات دست یابیم.

در این پایان نامه روشی به منظور جای دهی اولیه ماشین های مجازی به همراه اعمال کنترلی در انتخاب ماشین مجازی به منظور مهاجرت در نظر گرفته شده است. روش پیشنهاد شده از طریق شبیه ساز کلودسیم مورد بررسی و ارزیابی قرار گرفته است. نتایج آزمایشات نشان می دهد که اعمال روش مناسب در جای دهی و کنترل کردن مهاجرت به منظور جلوگیری از مهاجرت اضافی می تواند ما را در دست یافتن به اهدافی مانند بهبود توان و کیفیت سرویس کمک کند.

برای این منظور، از جمله کارهایی که در آینده بیشتر تمایل داریم به آن ها توجه کنیم، می توانیم به تکنیک های مربوط زمانبندی که در کاهش زمان اجرای برنامه تاثیر گذار است اشاره کنیم. همچنین با اعمال پارامتر های مربوط به هزینه ها و اعمال دستگاه های خنک کننده می توانیم در کاهش هزینه ها نیز بکوشیم.

مراجع

- [1] Beloglazov, A., & Buyya, R. (2013). "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers." *Concurrency and Computation: Practice and Experience*, 24(13), 1397-1420.
- [2] Bhaskar, R., & Shylaja, B. S. (2016). KNOWLEDGE BASED REDUCTION TECHNIQUE FOR VIRTUAL MACHINE PROVISIONING IN CLOUD COMPUTING. *International Journal of Computer Science and Information Security*, 14(7), 472.
- [3] Chien, N. K., Dong, V. S. G., Son, N. H., & Loc, H. D. (2016, March). An Efficient Virtual Machine Migration Algorithm Based on Minimization of Migration in Cloud Computing. In *International Conference on Nature of Computation and Communication* (pp. 62-71). Springer International Publishing.
- [4] Choi, H., Lim, J., Yu, H., & Lee, E. (2016). Task Classification Based Energy-Aware Consolidation in Clouds. *Scientific Programming*, 2016.
- [5] Da Cunha Rodrigues, G., Calheiros, R. N., Guimaraes, V. T., Santos, G. L. D., de Carvalho, M. B., Granville, L. Z., ... & Buyya, R. (2016, April). Monitoring of cloud computing environments: concepts, solutions, trends, and future directions. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (pp. 378-383). ACM.
- [6] Duan, H., Chen, C., Min, G., & Wu, Y. (2016). Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Future Generation Computer Systems*.
- [7] García-Valls, M., Cucinotta, T., & Lu, C. (2014). Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9), 726-740.
- [8] Goudarzi, H., & Pedram, M. (2016). Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter.
- [9] Höfer, C. N., & Karagiannis, G. (2011). Cloud computing services: taxonomy and comparison. *Journal of Internet Services and Applications*, 2(2), 81-94.

- [10] Ismaeel, S., Miri, A., & Al-Khazraji, A. (2016, March). Energy-consumption clustering in cloud data centre. In 2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC) (pp. 1-6). IEEE.
- [11] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.
- [12] Patel, R., Patel, H., & Patel, S. (2015). Quality of Service Based Efficient Resource Allocation in Cloud Computing, International Journal For Technological Research In Engineering Volume 2, Issue 9.
- [13] Raju, I. R. K., Varma, P. S., Sundari, M. R., & Moses, G. J. (2016). Deadline aware two stage scheduling algorithm in cloud computing. Indian Journal of Science and Technology, 9(4).
- [14] Rittinghouse, J. W., & Ransome, J. F. (2016). Cloud computing: implementation, management, and security. CRC press.
- [15] Subramanian, M., Bodge, A., & Pattabhi, R. (2016). U.S. Patent No. 20,160,019,265. Washington, DC: U.S. Patent and Trademark Office.
- [16] Tani, H. G., & El Amrani, C. (2016). Cloud Computing CPU Allocation and Scheduling Algorithms using CloudSim Simulator. International Journal of Electrical and Computer Engineering, 6(4), 1866.
- [17] Yang, M., Li, Y., Jin, D., Zeng, L., Wu, X., & Vasilakos, A. V. (2015). Software-defined and virtualized future mobile and wireless networks: A survey. Mobile Networks and Applications, 20(1), 4-18.

پیوست‌ها

کد استفاده شده در روش Basic Method(B_M)

```
1 package org.cloudbus.cloudsim.examples.power;
2
3 /*
4  * Title:          CloudSim Toolkit
5  * Description:    CloudSim (Cloud Simulation) Toolkit for Modeling
6                   and Simulation
7  * of Clouds
8  * Licence:        GPL — http://www.gnu.org/copyleft/gpl.html
9  *
10 * Copyright (c) 2009, The University of Melbourne, Australia
11 */
12 //import pso.*;
13 import java.text.DecimalFormat;
14 import java.util.ArrayList;
15 import java.util.Calendar;
16 import java.util.LinkedList;
17 import java.util.List;
18 import java.util.Map.Entry;
19
20 import org.cloudbus.cloudsim.Cloudlet;
21 import org.cloudbus.cloudsim.CloudletSchedulerDynamicWorkload;
22 import org.cloudbus.cloudsim.DatacenterBroker;
23 import org.cloudbus.cloudsim.DatacenterCharacteristics;
24 import org.cloudbus.cloudsim.Log;
25 import org.cloudbus.cloudsim.Storage;
```

```

26 import org.cloudbus.cloudsim.UtilizationModelStochastic;
27 import org.cloudbus.cloudsim.Vm;
28 import org.cloudbus.cloudsim.VmSchedulerTimeShared;
29 import org.cloudbus.cloudsim.core.CloudSim;
30 import org.cloudbus.cloudsim.power.PowerDatacenter;
31 import org.cloudbus.cloudsim.power.PowerVm;
32 import org.cloudbus.cloudsim.power.PowerHost;
33 //import org.cloudbus.cloudsim.power.PowerPe;
34 import org.cloudbus.cloudsim.power.models.PowerModelLinear;
35 import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
36 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
37 import org.cloudbus.cloudsim.File;
38
39
40 import java.util.ArrayList;
41 import java.util.List;
42 import java.util.LinkedList;
43 import java.io.BufferedReader;
44 import java.io.FileInputStream;
45 import java.io.InputStreamReader;
46 //import org.apache.commons.collections4.CollectionUtils;
47 import java.io.*;
48
49 //import cs.umu.se.vmp.data.DataGenerator;
50
51 import java.util.Vector;
52
53 import java.text.DecimalFormat;
54 import java.util.ArrayList;
55 import java.util.Calendar;

```

```

56 import java.util.LinkedList;
57 import java.util.List;
58
59 import java.util.ArrayList;
60 import java.util.HashMap;
61 import java.util.HashSet;
62 import java.util.LinkedList;
63 import java.util.List;
64 import java.util.Map;
65 import java.util.Set;
66
67
68 import java.util.ArrayList;
69 import java.util.List;
70 import java.util.LinkedList;
71 import java.io.BufferedReader;
72 import java.io.FileInputStream;
73 import java.io.InputStreamReader;
74 //import org.apache.commons.collections.CollectionUtils;
75 import java.io.*;
76
77 //import cs.umu.se.vmp.schema.PlacementRequest;
78 //import cs.umu.se.vmp.data.DataGenerator;
79
80 import java.util.Vector;
81
82 import java.text.DecimalFormat;
83 import java.util.ArrayList;
84 import java.util.Calendar;
85 import java.util.LinkedList;

```

```
86 import java.util.List;
87
88 import java.util.ArrayList;
89 import java.util.HashMap;
90 import java.util.HashSet;
91 import java.util.LinkedList;
92 import java.util.List;
93 import java.util.Map;
94 import java.util.Set;
95
96
97 import java.io.BufferedWriter;
98
99 import java.io.FileWriter;
100 import java.io.IOException;
101 import java.text.DecimalFormat;
102 import java.util.ArrayList;
103 import java.util.HashMap;
104 import java.util.LinkedList;
105 import java.util.List;
106 import java.util.Map;
107 import java.util.Scanner;
108 import org.cloudbus.cloudsim.Cloudlet;
109 import org.cloudbus.cloudsim.CloudletSchedulerDynamicWorkload;
110 import org.cloudbus.cloudsim.Datacenter;
111 import org.cloudbus.cloudsim.DatacenterBroker;
112 import org.cloudbus.cloudsim.DatacenterCharacteristics;
113 import org.cloudbus.cloudsim.Host;
114 import org.cloudbus.cloudsim.HostDynamicWorkload;
115 import org.cloudbus.cloudsim.HostStateHistoryEntry;
```

```

116 import org.cloudbus.cloudsim.Log;
117 import org.cloudbus.cloudsim.Pe;
118 import org.cloudbus.cloudsim.Storage;
119 import org.cloudbus.cloudsim.Vm;
120 import org.cloudbus.cloudsim.VmAllocationPolicy;
121 import org.cloudbus.cloudsim.VmSchedulerTimeSharedOverSubscription;
122 import org.cloudbus.cloudsim.VmStateHistoryEntry;
123 import org.cloudbus.cloudsim.power.PowerDatacenter;
124 import org.cloudbus.cloudsim.power.PowerDatacenterBroker;
125 import org.cloudbus.cloudsim.power.PowerVmAllocationPolicyAbstract;
126 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;
127 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationInterQuartileRange;
128 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegression;
129 import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;
130 import org.cloudbus.cloudsim.power.PowerHost;
131 import org.cloudbus.cloudsim.power.PowerHostUtilizationHistory;
132 import org.cloudbus.cloudsim.power.PowerVm;
133 import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
134 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
135 import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
136 import org.cloudbus.cloudsim.util.MathUtil;
137 import org.cloudbus.cloudsim.Host;
138 import org.cloudbus.cloudsim.Log;
139 import org.cloudbus.cloudsim.Vm;
140 import org.cloudbus.cloudsim.VmAllocationPolicy;
141 import org.cloudbus.cloudsim.core.CloudSim;
142 import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;

```

```

143 import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;
144 import org.cloudbus.cloudsim.Host;
145 import org.cloudbus.cloudsim.HostDynamicWorkload;
146 import org.cloudbus.cloudsim.Log;
147 import org.cloudbus.cloudsim.Vm;
148 import org.cloudbus.cloudsim.core.CloudSim;
149 import org.cloudbus.cloudsim.power.lists.PowerVmList;
150 import org.cloudbus.cloudsim.util.ExecutionTimeMeasurer;
151 import java.util.HashSet;
152 import org.cloudbus.cloudsim.UtilizationModelStochastic;
153 import org.cloudbus.cloudsim.examples.power.Helper;
154 import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;
155 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMaximumCorrelation;
156 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumMigrationTimebase;
157
158
159 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumUtilizationbase;
160 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyRandomSelection;
161 import org.cloudbus.cloudsim.CloudletSchedulerDynamicWorkload;
162 import org.cloudbus.cloudsim.Datacenter;
163 import org.cloudbus.cloudsim.DatacenterBroker;
164 import org.cloudbus.cloudsim.DatacenterCharacteristics;
165 import org.cloudbus.cloudsim.Host;
166 import org.cloudbus.cloudsim.HostDynamicWorkload;
167 import org.cloudbus.cloudsim.HostStateHistoryEntry;
168 import org.cloudbus.cloudsim.Log;

```



```

169 import org.cloudbus.cloudsim.Pe;
170 import org.cloudbus.cloudsim.Storage;
171 import org.cloudbus.cloudsim.Vm;
172 import org.cloudbus.cloudsim.VmAllocationPolicy;
173 import org.cloudbus.cloudsim.VmSchedulerTimeSharedOverSubscription;
174 import org.cloudbus.cloudsim.VmStateHistoryEntry;
175 import org.cloudbus.cloudsim.power.PowerDatacenter;
176 import org.cloudbus.cloudsim.power.PowerHostUtilizationHistory;
177
178 import org.cloudbus.cloudsim.power.PowerHost;
179 //import org.cloudbus.cloudsim.power.PowerVm;
180 import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
181 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
182 import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
183
184
185 import org.cloudbus.cloudsim.core.CloudSim;
186 import org.cloudbus.cloudsim.UtilizationModelStochastic;
187 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegression;
188 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationStaticThreshold;
189 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;
190 import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;
191 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumMigrationTime;
192 import org.cloudbus.cloudsim.VmAllocationPolicy;
193 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegressionRobu

```

```

194 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;
195 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationMedianAbsoluteDevia
196
197 import java.sql.*;
198 import java.text.DecimalFormat;
199 import java.util.ArrayList;
200 import java.util.Calendar;
201 import java.util.LinkedList;
202 import java.util.List;
203
204
205
206 /**
207  * An example of a power aware data center. In this example the
    placement of VMs
208  * is continuously adapted using VM migration in order to minimize
    the number
209  * of physical nodes in use, while idle nodes are switched off to
    save energy.
210  * The CPU utilization of each host is kept under the specified
    utilization threshold.
211  */
212 public class B_M {
213
214     /** The cloudlet list. */
215     private static List<Cloudlet> cloudletList;
216
217     /** The vm list. */

```

```

218     private static List<Vm> vmList;
219     //private static List<Vm> vmLists;
220
221     //private static double utilizationThreshold = 0.8;
222
223     //private static double hostsNumber =110;
224     // private static List<Vm> vmList;
225
226     //private static double utilizationThreshold = 0.8;
227
228     private static double hostsNumber = 100;
229     //private static double vmsNumber = 50;
230     private static double cloudletsNumber =100;
231     public static Vm[] vm=new PowerVm[100] ;
232     public static int requestnumber=100;
233     //private static double vmsNumber = 80;
234     //private static double cloudletsNumber =10;
235     //private static double vmsNumber = 10;//
236     //private static double cloudletsNumber = 20;///
237
238     /**
239     * Creates main() to run this example.
240     *
241     * @param args the args
242     */
243
244     public static Connection con;
245
246     //public static double[][] vmarray = new
        double[resource_num][5];

```

```

247 // public static double[][] cloudarray = new
        double[task_num][2];
248
249 // public static int[] vm_list = new int[task_num];
250 public static LinkedList<Vm> list = new LinkedList<Vm>();
251 public static LinkedList<Cloudlet> list2 = new
        LinkedList<Cloudlet>();
252
253 //public static Vm[] vm = new PowerVm[resource_num];
254 // public static Cloudlet[] cloudlet = new Cloudlet[task_num];
255 public static int t ;
256
257
258
259 private static List<Vm> createVmList(int brokerId ,int
        requestnumber) {
260 //requestnumber=50;
261 List<Vm> vmList = new ArrayList<Vm>();
262 for (int i = 0; i < requestnumber ; i++) {
263     int vmType = i / (int) Math.ceil((double)
        requestnumber / Constants.VM_TYPES);
264     vm[i]=new PowerVm(
265         i ,
266         brokerId ,
267         Constants.VM_MIPS[vmType] ,
268         Constants.VM_PES[vmType] ,
269         Constants.VM_RAM[vmType] ,
270         Constants.VM_BW,
271         Constants.VM_SIZE,
272         1,

```

```

273         "Xen",
274         new
                CloudletSchedulerDynamicWorkload( Constants.SCHEDULING_INTERVAL);
                Constants.VM_PES[vmType]) ,
275         Constants.SCHEDULING_INTERVAL);
276 //         wrapper r =new wrapper ();
277         vmlist.add(vm[ i ]);
278     }
279
280
281     return vmlist;
282     //int vmType = i / (int) Math.ceil((double)
                requestnumber / Constants.VM_TYPES);
283
284
285
286
287 }
288
289 private static List<Cloudlet> createCloudletList(int
        brokerId ) {
290 List<Cloudlet> list = new ArrayList<Cloudlet>();
291
292     long length = 150000; // 10 min on 250 MIPS
293     int pesNumber = 1;
294     long fileSize = 300;
295     long outputSize = 300;
296     // cloudletsNumber =50;
297     for (int i = 0; i < cloudletsNumber; i++) {

```

```

298         Cloudlet cloudlet = new Cloudlet(i, length,
        pesNumber, fileSize, outputSize, new
        UtilizationModelStochastic(), new
        UtilizationModelStochastic(), new
        UtilizationModelStochastic());
299         cloudlet.setUserId(brokerId);
300         cloudlet.setVmId(i);
301         list.add(cloudlet);
302     }
303
304     return list;
305
306 }
307
308
309
310 public static void main(String[] args) {
311
312
313
314
315     Vector v = new Vector();
316     int max = 0;
317
318     try {
319
320         int num_user = 1; // number of cloud users
321         Calendar calendar = Calendar.getInstance();
322         boolean trace_flag = false; // mean trace
        GridSim events

```

```

323
324         // Initialize the CloudSim library
325         CloudSim.init(num_user, calendar,
326                        trace_flag);
327
328         // Second step: Create Datacenters
329         // Datacenters are the resource providers in
330         // CloudSim. We need at
331         // list one of them to run a CloudSim
332         // simulation
333
334         PowerDatacenter datacenter =
335             (PowerDatacenter) createDatacenter(
336                 "Datacenter0",
337                 PowerDatacenter.class);
338
339
340
341
342
343
344         // Third step: Create Broker
345         DatacenterBroker broker = createBroker();
346         int brokerId = broker.getId();
347
348
349
350
351         // vmarray.add(vmList);
352
353
354
355
356
357
358         vmList=createVmList(brokerId ,requestnumber);
359         cloudletList = createCloudletList(brokerId);
360
361
362
363         broker.submitVmList(vmList);
364         broker.submitCloudletList(cloudletList);

```

```

349
350         // for(i=0; i<task_num; i++){
351         // broker.bindCloudletToVm( cloudlet[i].getCloudletId(),vm_l
352         // }
353
354
355
356
357
358         // Fourth step: Create one virtual machine
359         // vmList = createVms(brokerId);
360
361         // submit vm list to the broker
362         // broker.submitVmList(vmarray);
363
364         // Fifth step: Create one cloudlet
365         // broker.submitVmList(vmarray);
366         // cloudletList =
367             createCloudletList(brokerId);
368
369         // submit cloudlet list to the broker
370         // broker.submitCloudletList(cloudletList);
371
372         // Sixth step: Starts the simulation
373         double lastClock =
374             CloudSim.startSimulation();
375
376         // Final step: Print results when simulation
377         // is over

```



```

376         List<Cloudlet> newList =
            broker.getCloudletReceivedList();
377         Log.println("Received " + newList.size() +
            " cloudlets");
378
379         CloudSim.stopSimulation();
380
381         printCloudletList(datacenter,
382                             vmList, newList);
383
384
385
386         Log.println();
387         Log.println(String.format("Total
            simulation time: %.2f sec", lastClock));
388         Log.println(String.format("Energy
            consumption: %.2f kWh",
            datacenter.getPower() / (3600 * 1000)));
389         Log.println(String.format("Number of VM
            migrations: %d",
            datacenter.getMigrationCount()));
390         //Log.println(String.format("Number of SLA
            violations: %d", sla.size()));
391         //Log.println(String.format("SLA violation
            percentage: %.2f%%", (double) sla.size()
            * 100 / numberOfAllocations));
392         //Log.println(String.format("Average SLA
            violation: %.2f%%", averageSla));
393         Log.println();
394

```

```

395         } catch (Exception e) {
396             e.printStackTrace();
397             Log.println("Unwanted errors happen");
398         }
399
400         Log.println("SingleThreshold finished!");
401     }
402
403     /**
404      * Creates the cloudlet list.
405      *
406      * @param brokerId the broker id
407      *
408      * @return the cloudlet list
409      */
410
411
412
413     /**
414      * Creates the vms.
415      *
416      * @param brokerId the broker id
417      *
418      * @return the list < vm>
419      */
420
421
422
423
424     private static Datacenter createDatacenter(String name,

```

```

425         Class<? extends Datacenter> datacenterClass)
426         throws Exception {
427     // Here are the steps needed to create a
428         PowerDatacenter:
429     // 1. We need to create an object of HostList2 to
430         store
431     // our machine
432     List<PowerHost> hostList = new
433         ArrayList<PowerHost>();
434
435     //Class<? extends Datacenter> datacenterClass;
436     double maxPower = 250; // 250W
437     double staticPowerPercent = 0.7; // 70%
438
439     int[] mips = { 5000, 5500 };
440     int [] ram = {4096, 4096}; // host memory (MB)
441     long storage = 10000; // host storage
442     int bw = 10000;
443
444     int[] HOST_MIPS = { 5000, 4500 };
445     int[] HOST_PES      = { 1, 1 };
446     for (int i = 0; i < hostsNumber; i++) {
447         int hostType = i % Constants.HOST_TYPES;
448
449         List<Pe> peList = new ArrayList<Pe>();
450         for (int j = 0; j <
451             Constants.HOST_PES[hostType]; j++) {
452             peList.add(new Pe(j, new
453                 PeProvisionerSimple(HOST_MIPS[hostType])));
454         }
455
456         hostList.add(new PowerHostUtilizationHistory(

```

```

450         i ,
451         new
            RamProvisionerSimple(ram[hostType]) ,
452         new
            BwProvisionerSimple( Constants.HOST_BW) ,
453         Constants.HOST_STORAGE,
454         peList ,
455         new
            VmSchedulerTimeShared( peList) ,
456         Constants.HOST_POWER[hostType]) ); //
            This is our machine
457     }
458
459     // 5. Create a DatacenterCharacteristics object that
        stores the
460     // properties of a Grid resource: architecture , OS,
        list of
461     // Machines , allocation policy: time- or
        space-shared , time zone
462     // and its price (G$/PowerPe time unit).
463     String arch = "x86"; // system architecture
464     String os = "Linux"; // operating system
465     String vmm = "Xen";
466     double time_zone = 10.0; // time zone this resource
        located
467     double cost = 3.0; // the cost of using processing
        in this resource
468     double costPerMem = 0.05; // the cost of using
        memory in this resource

```

```

469         double costPerStorage = 0.001; // the cost of using
           storage in this resource
470         double costPerBw = 0.0; // the cost of using bw in
           this resource
471         String vmAllocationPolicyName="mad";
472             String vmSelectionPolicyName= "mmt";
473             String parameterName="2.5";
474             VmAllocationPolicy vmAllocationPolicy = null;
475         PowerVmSelectionPolicy vmSelectionPolicy = null;
476         if (!vmSelectionPolicyName.isEmpty()) {
477             vmSelectionPolicy =
478                 getVmSelectionPolicy(vmSelectionPolicyName);
479         }
480         double parameter = 0;
481         if (!parameterName.isEmpty()) {
482             parameter = Double.valueOf(parameterName);
483         }
484
485         if (vmAllocationPolicyName.equals("iqr")) {
486             PowerVmAllocationPolicyMigrationAbstract
487                 fallbackVmSelectionPolicy = new
488                     PowerVmAllocationPolicyMigrationStaticThreshold(
489                         hostList ,
490                         vmSelectionPolicy ,
491                         0.7);
492             vmAllocationPolicy = new
493                 PowerVmAllocationPolicyMigrationInterQuartileRange(
494                     hostList ,
495                     vmSelectionPolicy ,

```

```

493             parameter ,
494             fallbackVmSelectionPolicy );
495 } else if (vmAllocationPolicyName.equals("mad")) {
496     PowerVmAllocationPolicyMigrationAbstract
497         fallbackVmSelectionPolicy = new
498         PowerVmAllocationPolicyMigrationStaticThreshold(
499             hostList ,
500             vmSelectionPolicy ,
501             0.7);
502 vmAllocationPolicy = new
503     PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation
504         hostList ,
505         vmSelectionPolicy ,
506         parameter ,
507         fallbackVmSelectionPolicy );
508 } else if (vmAllocationPolicyName.equals("lr")) {
509     PowerVmAllocationPolicyMigrationAbstract
510         fallbackVmSelectionPolicy = new
511         PowerVmAllocationPolicyMigrationStaticThreshold(
512             hostList ,
513             vmSelectionPolicy ,
514             0.7);
515 vmAllocationPolicy = new
516     PowerVmAllocationPolicyMigrationLocalRegression(
517         hostList ,
518         vmSelectionPolicy ,
519         parameter ,
520         Constants.SCHEDULING_INTERVAL,
521         fallbackVmSelectionPolicy );
522 } else if (vmAllocationPolicyName.equals("lrr")) {

```

```

517         PowerVmAllocationPolicyMigrationAbstract
            fallbackVmSelectionPolicy = new
            PowerVmAllocationPolicyMigrationStaticThreshold(
518                 hostList ,
519                 vmSelectionPolicy ,
520                 0.7);
521     vmAllocationPolicy = new
            PowerVmAllocationPolicyMigrationLocalRegressionRobust(
522                 hostList ,
523                 vmSelectionPolicy ,
524                 parameter ,
525                 Constants.SCHEDULING_INTERVAL,
526                 fallbackVmSelectionPolicy);
527 } else if (vmAllocationPolicyName.equals("thr")) {
528     vmAllocationPolicy = new
            PowerVmAllocationPolicyMigrationStaticThreshold(
529                 hostList ,
530                 vmSelectionPolicy ,
531                 parameter);
532 } else if (vmAllocationPolicyName.equals("dvfs")) {
533     vmAllocationPolicy = new
            PowerVmAllocationPolicySimple(hostList);
534 } else {
535     System.out.println("Unknown VM allocation
            policy: " + vmAllocationPolicyName);
536     System.exit(0);
537 }
538
539

```

```

540         DatacenterCharacteristics characteristics = new
            DatacenterCharacteristics(
541             arch, os, vmm, hostList, time_zone,
                cost, costPerMem, costPerStorage,
                costPerBw);
542
543         // 6. Finally, we need to create a PowerDatacenter
            object.
544         Datacenter datacenter = null;
545         try {
546             datacenter = datacenterClass.getConstructor(
547                 String.class,
548                 DatacenterCharacteristics.class,
549                 VmAllocationPolicy.class,
550                 List.class,
551                 Double.TYPE).newInstance(
552                     name,
553                     characteristics,
554                     vmAllocationPolicy,
555                     new LinkedList<Storage>(),
556                     Constants.SCHEDULING_INTERVAL);
557         } catch (Exception e) {
558             e.printStackTrace();
559         }
560
561         return datacenter;
562     }
563     protected static PowerVmSelectionPolicy getVmSelectionPolicy(String
        vmSelectionPolicyName) {
564         PowerVmSelectionPolicy vmSelectionPolicy = null;

```



```

565
566         if (vmSelectionPolicyName.equals("mc")) {
567             vmSelectionPolicy = new
568                 PowerVmSelectionPolicyMaximumCorrelation(
569                     new
570                         PowerVmSelectionPolicyMinimumMigrationTimebase();
571         } else if (vmSelectionPolicyName.equals("mmt")) {
572             vmSelectionPolicy = new
573                 PowerVmSelectionPolicyMinimumMigrationTimebase();
574         } else if (vmSelectionPolicyName.equals("mu")) {
575             vmSelectionPolicy = new
576                 PowerVmSelectionPolicyMinimumUtilizationbase();
577         } else if (vmSelectionPolicyName.equals("rs")) {
578             vmSelectionPolicy = new
579                 PowerVmSelectionPolicyRandomSelection();
580         } else {
581             System.out.println("Unknown VM selection
582                 policy: " + vmSelectionPolicyName);
583             System.exit(0);
584         }
585         return vmSelectionPolicy;
586     }
587
588     // We strongly encourage users to develop their own broker
589     // policies , to
590     // submit vms and cloudlets according
591     // to the specific rules of the simulated scenario
592     /**
593      * Creates the broker.
594      *
595      * @return the datacenter broker

```

```

588      */
589      private static DatacenterBroker createBroker() {
590          DatacenterBroker broker = null;
591          try {
592              broker = new DatacenterBroker("Broker");
593          } catch (Exception e) {
594              e.printStackTrace();
595              return null;
596          }
597          return broker;
598      }
599
600      /**
601       * Prints the Cloudlet objects.
602       *
603       * @param list list of Cloudlets
604       */
605      private static void printCloudletList(PowerDatacenter
        datacenter ,
606          List<Vm> vms, List<Cloudlet> list) {
607          int size = list.size();
608          List<Host> hosts = datacenter.getHostList();
609
610          int numberOfHosts = hosts.size();
611          int numberOfVms = vms.size();
612          Map<String , Double> slaMetrics = getSlaMetrics(vms);
613          double slaOverall = slaMetrics.get("overall");
614          double slaAverage = slaMetrics.get("average");
615          double slaDegradationDueToMigration =
            slaMetrics.get("underallocated_migration");

```

```

616         double slaTimePerActiveHost =
            getSlaTimePerActiveHost( hosts );
617
618         double sla = slaTimePerActiveHost *
            slaDegradationDueToMigration;
619         List<Double> timeBeforeHostShutdown =
            getTimesBeforeHostShutdown( hosts );
620
621         int numberOfHostShutdowns =
            timeBeforeHostShutdown.size();
622
623         Cloudlet cloudlet;
624
625         String indent = "\t";
626         Log.println();
627
628         Log.println();
629         Log.println( String.format( "%d",
            numberOfHosts ));
630         Log.println( String.format( "%d",
            numberOfVms ));
631         Log.println( String.format( "SLA: %.5f%%",
            sla * 100 ));
632         Log.println( String.format( "SLA time per
            active host: %.2f%%",
            slaTimePerActiveHost * 100 ));
633         Log.println( String.format( "SLA perf
            degradation due to migration: %.2f%%",
            slaDegradationDueToMigration
            * 100 ));
634

```

```

635         Log.println( String.format("Overall SLA
           violation: %.2f%%", slaOverall * 100));
636         Log.println( String.format("Average SLA
           violation: %.2f%%", slaAverage * 100));
637         Log.println( String.format("Number of host
           shutdowns: %d", numberOfHostShutdowns));
638
639     }
640     protected static Map<String , Double> getSlaMetrics(List<Vm>
        vms) {
641         Map<String , Double> metrics = new HashMap<String ,
            Double>();
642         List<Double> slaViolation = new LinkedList<Double>();
643         double totalAllocated = 0;
644         double totalRequested = 0;
645         double totalUnderAllocatedDueToMigration = 0;
646
647         for (Vm vm : vms) {
648             double vmTotalAllocated = 0;
649             double vmTotalRequested = 0;
650             double vmUnderAllocatedDueToMigration = 0;
651             double previousTime = -1;
652             double previousAllocated = 0;
653             double previousRequested = 0;
654             boolean previousIsInMigration = false;
655
656             for (VmStateHistoryEntry entry :
                vm.getStateHistory()) {
657                 if (previousTime != -1) {

```

```

658         double timeDiff =
            entry.getTime() -
            previousTime;
659         vmTotalAllocated +=
            previousAllocated *
            timeDiff;
660         vmTotalRequested +=
            previousRequested *
            timeDiff;
661
662         if (previousAllocated <
            previousRequested) {
663             slaViolation.add((previousRequested
                -
                previousAllocated)
                /
                previousRequested);
664         if
            (previousIsInMigration)
            {
665             vmUnderAllocatedDueToMigra
                +=
                (previousRequested
                -
                previousAllocated)
                *
                timeDiff;
666             }
667         }
668     }
669 }

```

```

670
671         previousAllocated =
672             entry.getAllocatedMips();
673         previousRequested =
674             entry.getRequestMips();
675         previousTime = entry.getTime();
676         previousIsInMigration =
677             entry.isInMigration();
678     }
679
680     totalAllocated += vmTotalAllocated;
681     totalRequested += vmTotalRequested;
682     totalUnderAllocatedDueToMigration +=
683         vmUnderAllocatedDueToMigration;
684 }
685
686 metrics.put("overall", (totalRequested -
687     totalAllocated) / totalRequested);
688 if (slaViolation.isEmpty()) {
689     metrics.put("average", 0.);
690 } else {
691     metrics.put("average",
692         MathUtil.mean(slaViolation));
693 }
694
695 metrics.put("underallocated_migration",
696     totalUnderAllocatedDueToMigration /
697     totalRequested);
698 // metrics.put("sla_time_per_vm_with_migration",
699     slaViolationTimePerVmWithMigration /
700     totalTime);

```

```

691         // metrics.put("sla_time_per_vm_without_migration",
        // slaViolationTimePerVmWithoutMigration /
692         // totalTime);
693
694         return metrics;
695     }
696     protected static double getSlaTimePerActiveHost(List<Host>
        hosts) {
697         double slaViolationTimePerHost = 0;
698         double totalTime = 0;
699
700         for (Host _host : hosts) {
701             HostDynamicWorkload host =
                (HostDynamicWorkload) _host;
702             double previousTime = -1;
703             double previousAllocated = 0;
704             double previousRequested = 0;
705             boolean previousIsActive = true;
706
707             for (HostStateHistoryEntry entry :
                host.getStateHistory()) {
708                 if (previousTime != -1 &&
                    previousIsActive) {
709                     double timeDiff =
                        entry.getTime() -
                            previousTime;
710                     totalTime += timeDiff;
711                     if (previousAllocated <
                        previousRequested) {

```

```

712                                     slaViolationTimePerHost
                                     += timeDiff;
713                                 }
714                             }
715
716                             previousAllocated =
                                entry.getAllocatedMips();
717                             previousRequested =
                                entry.getRequestMips();
718                             previousTime = entry.getTime();
719                             previousIsActive = entry.isActive();
720                         }
721                     }
722
723                     return slaViolationTimePerHost / totalTime;
724 }
725 public static List<Double>
    getTimesBeforeHostShutdown(List<Host> hosts) {
726     List<Double> timeBeforeShutdown = new
        LinkedList<Double>();
727     for (Host host : hosts) {
728         boolean previousIsActive = true;
729         double lastTimeSwitchedOn = 0;
730         for (HostStateHistoryEntry entry :
            ((HostDynamicWorkload)
            host).getStateHistory()) {
731             if (previousIsActive == true &&
                entry.isActive() == false) {
732                 timeBeforeShutdown.add(entry.getTime()
                    - lastTimeSwitchedOn);

```



```

733         }
734         if (previousIsActive == false &&
            entry.isActive() == true) {
735             lastTimeSwitchedOn =
                entry.getTime();
736         }
737         previousIsActive = entry.isActive();
738     }
739 }
740 return timeBeforeShutdown;
741 }
742
743
744
745
746 }

```

کد استفاده شده در روش S_S (Suggested Solution)

```

1 package org.cloudbus.cloudsim.examples.power;
2
3 /*
4  * Title:          CloudSim Toolkit
5  * Description:    CloudSim (Cloud Simulation) Toolkit for Modeling
6  *                and Simulation
7  *                of Clouds
8  * Licence:        GPL — http://www.gnu.org/copyleft/gpl.html

```

```

8  *
9  * Copyright (c) 2009, The University of Melbourne, Australia
10 */
11
12 //import pso.*;
13 import java.text.DecimalFormat;
14 import java.util.ArrayList;
15 import java.util.Calendar;
16 import java.util.LinkedList;
17 import java.util.List;
18 import java.util.Map.Entry;
19
20 import org.cloudbus.cloudsim.Cloudlet;
21 import org.cloudbus.cloudsim.CloudletSchedulerDynamicWorkload;
22 import org.cloudbus.cloudsim.DatacenterBroker;
23 import org.cloudbus.cloudsim.DatacenterCharacteristics;
24 import org.cloudbus.cloudsim.Log;
25 import org.cloudbus.cloudsim.Storage;
26 import org.cloudbus.cloudsim.UtilizationModelStochastic;
27 import org.cloudbus.cloudsim.Vm;
28 import org.cloudbus.cloudsim.VmSchedulerTimeShared;
29 import org.cloudbus.cloudsim.core.CloudSim;
30 import org.cloudbus.cloudsim.power.PowerDatacenter;
31 import org.cloudbus.cloudsim.power.PowerVm;
32 import org.cloudbus.cloudsim.power.PowerHost;
33 //import org.cloudbus.cloudsim.power.PowerPe;
34 import org.cloudbus.cloudsim.power.models.PowerModelLinear;
35 import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
36 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
37 import org.cloudbus.cloudsim.File;

```

```
38
39
40 import java.util.ArrayList;
41 import java.util.List;
42 import java.util.LinkedList;
43 import java.io.BufferedReader;
44 import java.io.FileInputStream;
45 import java.io.InputStreamReader;
46 //import org.apache.commons.collections4.CollectionUtils;
47 import java.io.*;
48
49 //import cs.umu.se.vmp.data.DataGenerator;
50
51 import java.util.Vector;
52
53 import java.text.DecimalFormat;
54 import java.util.ArrayList;
55 import java.util.Calendar;
56 import java.util.LinkedList;
57 import java.util.List;
58
59 import java.util.ArrayList;
60 import java.util.HashMap;
61 import java.util.HashSet;
62 import java.util.LinkedList;
63 import java.util.List;
64 import java.util.Map;
65 import java.util.Set;
66
67
```

```

68 import java.util.ArrayList;
69 import java.util.List;
70 import java.util.LinkedList;
71 import java.io.BufferedReader;
72 import java.io.FileInputStream;
73 import java.io.InputStreamReader;
74 //import org.apache.commons.collections.CollectionUtils;
75 import java.io.*;
76
77 //import cs.umu.se.vmp.schema.PlacementRequest;
78 //import cs.umu.se.vmp.data.DataGenerator;
79
80 import java.util.Vector;
81
82 import java.text.DecimalFormat;
83 import java.util.ArrayList;
84 import java.util.Calendar;
85 import java.util.LinkedList;
86 import java.util.List;
87
88 import java.util.ArrayList;
89 import java.util.HashMap;
90 import java.util.HashSet;
91 import java.util.LinkedList;
92 import java.util.List;
93 import java.util.Map;
94 import java.util.Set;
95
96
97 import java.io.BufferedWriter;

```

```

98
99 import java.io.FileWriter;
100 import java.io.IOException;
101 import java.text.DecimalFormat;
102 import java.util.ArrayList;
103 import java.util.HashMap;
104 import java.util.LinkedList;
105 import java.util.List;
106 import java.util.Map;
107 import java.util.Scanner;
108 import org.cloudbus.cloudsim.Cloudlet;
109 import org.cloudbus.cloudsim.CloudletSchedulerDynamicWorkload;
110 import org.cloudbus.cloudsim.Datacenter;
111 import org.cloudbus.cloudsim.DatacenterBroker;
112 import org.cloudbus.cloudsim.DatacenterCharacteristics;
113 import org.cloudbus.cloudsim.Host;
114 import org.cloudbus.cloudsim.HostDynamicWorkload;
115 import org.cloudbus.cloudsim.HostStateHistoryEntry;
116 import org.cloudbus.cloudsim.Log;
117 import org.cloudbus.cloudsim.Pe;
118 import org.cloudbus.cloudsim.Storage;
119 import org.cloudbus.cloudsim.Vm;
120 import org.cloudbus.cloudsim.VmAllocationPolicy;
121 import org.cloudbus.cloudsim.VmSchedulerTimeSharedOverSubscription;
122 import org.cloudbus.cloudsim.VmStateHistoryEntry;
123 import org.cloudbus.cloudsim.power.PowerDatacenter;
124 import org.cloudbus.cloudsim.power.PowerDatacenterBroker;
125 import org.cloudbus.cloudsim.power.PowerVmAllocationPolicyAbstract;
126 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;

```

```

127 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationInterQuartileRange;
128 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegression;
129 import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;
130 import org.cloudbus.cloudsim.power.PowerHost;
131 import org.cloudbus.cloudsim.power.PowerHostUtilizationHistory;
132 import org.cloudbus.cloudsim.power.PowerVm;
133 import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
134 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
135 import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
136 import org.cloudbus.cloudsim.util.MathUtil;
137 import org.cloudbus.cloudsim.Host;
138 import org.cloudbus.cloudsim.Log;
139 import org.cloudbus.cloudsim.Vm;
140 import org.cloudbus.cloudsim.VmAllocationPolicy;
141 import org.cloudbus.cloudsim.core.CloudSim;
142 import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;
143 import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;
144 import org.cloudbus.cloudsim.Host;
145 import org.cloudbus.cloudsim.HostDynamicWorkload;
146 import org.cloudbus.cloudsim.Log;
147 import org.cloudbus.cloudsim.Vm;
148 import org.cloudbus.cloudsim.core.CloudSim;
149 import org.cloudbus.cloudsim.power.lists.PowerVmList;
150 import org.cloudbus.cloudsim.util.ExecutionTimeMeasurer;
151 import java.util.HashSet;
152 import org.cloudbus.cloudsim.UtilizationModelStochastic;
153 import org.cloudbus.cloudsim.examples.power.Helper;
154 import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;

```

```

155 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMaximumCorrelation;
156 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumMigrationTime;
157
158
159 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumUtilization;
160 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyRandomSelection;
161 import org.cloudbus.cloudsim.CloudletSchedulerDynamicWorkload;
162 import org.cloudbus.cloudsim.Datacenter;
163 import org.cloudbus.cloudsim.DatacenterBroker;
164 import org.cloudbus.cloudsim.DatacenterCharacteristics;
165 import org.cloudbus.cloudsim.Host;
166 import org.cloudbus.cloudsim.HostDynamicWorkload;
167 import org.cloudbus.cloudsim.HostStateHistoryEntry;
168 import org.cloudbus.cloudsim.Log;
169 import org.cloudbus.cloudsim.Pe;
170 import org.cloudbus.cloudsim.Storage;
171 import org.cloudbus.cloudsim.Vm;
172 import org.cloudbus.cloudsim.VmAllocationPolicy;
173 import org.cloudbus.cloudsim.VmSchedulerTimeSharedOverSubscription;
174 import org.cloudbus.cloudsim.VmStateHistoryEntry;
175 import org.cloudbus.cloudsim.power.PowerDatacenter;
176 import org.cloudbus.cloudsim.power.PowerHostUtilizationHistory;
177
178 import org.cloudbus.cloudsim.power.PowerHost;
179 //import org.cloudbus.cloudsim.power.PowerVm;
180 import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

```

```

181 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
182 import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
183
184
185 import org.cloudbus.cloudsim.core.CloudSim;
186 import org.cloudbus.cloudsim.UtilizationModelStochastic;
187 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegression;
188 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationStaticThreshold;
189 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;
190 import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;
191 import
    org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumMigrationTime;
192 import org.cloudbus.cloudsim.VmAllocationPolicy;
193 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegressionRobu
194 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;
195 import
    org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationMedianAbsoluteDevia
196
197 import java.sql.*;
198 import java.text.DecimalFormat;
199 import java.util.ArrayList;
200 import java.util.Calendar;
201 import java.util.LinkedList;
202 import java.util.List;
203

```



```

204
205
206 /**
207  * An example of a power aware data center. In this example the
208    placement of VMs
209  * is continuously adapted using VM migration in order to minimize
210    the number
211  * of physical nodes in use, while idle nodes are switched off to
212    save energy.
213  * The CPU utilization of each host is kept under the specified
214    utilization threshold.
215  */
216 public class S_S {
217
218     /** The cloudlet list. */
219     private static List<Cloudlet> cloudletList;
220
221     /** The vm list. */
222     private static List<Vm> vmList;
223     //private static List<Vm> vmLists;
224
225     //private static double utilizationThreshold = 0.8;
226
227     //private static double hostsNumber =110;
228     // private static List<Vm> vmList;
229
230     //private static double utilizationThreshold = 0.8;
231
232     private static double hostsNumber = 100;
233     //private static double vmsNumber = 50;

```

```

230     private static double cloudletsNumber =100;
231     public static Vm[] vm=new PowerVm[100] ;
232     public static int requestnumber=100;
233     //private static double vmsNumber = 80;
234     //private static double cloudletsNumber =10;
235     //private static double vmsNumber = 10;//
236     //private static double cloudletsNumber = 20;//
237
238     /**
239      * Creates main() to run this example.
240      *
241      * @param args the args
242      */
243
244     public static Connection con;
245
246     //public static double[][] vmarray = new
247         double[resource_num][5];
248
249     // public static double[][] cloudarray = new
250         double[task_num][2];
251
252     // public static int[] vm_list = new int[task_num];
253     public static LinkedList<Vm> list = new LinkedList<Vm>();
254     public static LinkedList<Cloudlet> list2 = new
255         LinkedList<Cloudlet>();
256
257     //public static Vm[] vm = new PowerVm[resource_num];
258     // public static Cloudlet[] cloudlet = new Cloudlet[task_num];
259     public static int t ;

```

```

257
258
259     private static List<Vm> createVmList(int brokerId, int
        requestnumber) {
260         //requestnumber=50;
261         List<Vm> vmlist = new ArrayList<Vm>();
262         for (int i = 0; i < requestnumber ; i++) {
263             int vmType = i / (int) Math.ceil((double)
                requestnumber / Constants.VM_TYPES);
264             vm[i]=new PowerVm(
265                 i,
266                 brokerId,
267                 Constants.VM_MIPS[vmType],
268                 Constants.VM_PES[vmType],
269                 Constants.VM_RAM[vmType],
270                 Constants.VM_BW,
271                 Constants.VM_SIZE,
272                 1,
273                 "Xen",
274                 new
                    CloudletSchedulerDynamicWorkload( Constants
                        Constants.VM_PES[vmType]),
                    Constants.SCHEDULING_INTERVAL);
275
276         //         wrapper    r =new wrapper ();
277         vmlist.add(vm[i]);
278     }
279
280
281     return vmlist;

```

```

282         //int vmType = i / (int) Math.ceil((double)
           requestnumber / Constants.VM_TYPES);
283
284
285
286
287     }
288
289     private static List<Cloudlet> createCloudletList(int
           brokerId ) {
290         List<Cloudlet> list = new ArrayList<Cloudlet>();
291
292         long length = 150000; // 10 min on 250 MIPS
293         int pesNumber = 1;
294         long fileSize = 300;
295         long outputSize = 300;
296         // cloudletsNumber =50;
297         for (int i = 0; i < cloudletsNumber; i++) {
298             Cloudlet cloudlet = new Cloudlet(i, length,
                pesNumber, fileSize, outputSize, new
                UtilizationModelStochastic(), new
                UtilizationModelStochastic(), new
                UtilizationModelStochastic());
299             cloudlet.setUserId(brokerId);
300             cloudlet.setVmId(i);
301             list.add(cloudlet);
302         }
303
304         return list;
305

```

```

306     }
307
308
309
310     public static void main(String[] args) {
311
312
313
314
315         Vector v = new Vector();
316         int max = 0;
317
318         try {
319
320             int num_user = 1; // number of cloud users
321             Calendar calendar = Calendar.getInstance();
322             boolean trace_flag = false; // mean trace
323                                     GridSim events
324
325             // Initialize the CloudSim library
326             CloudSim.init(num_user, calendar,
327                           trace_flag);
328
329             // Second step: Create Datacenters
330             // Datacenters are the resource providers in
331             // CloudSim. We need at
332             // list one of them to run a CloudSim
333             // simulation
334             PowerDatacenter datacenter =
335                 (PowerDatacenter) createDatacenter(

```

```

331         "Datacenter0",
332         PowerDatacenter.class);
333
334
335
336     // Third step: Create Broker
337     DatacenterBroker broker = createBroker();
338     int brokerId = broker.getId();
339
340
341     // vmarray.add(vmList);
342
343
344     vmList=createVmList(brokerId,requestnumber);
345     cloudletList = createCloudletList(brokerId);
346
347     broker.submitVmList(vmList);
348     broker.submitCloudletList(cloudletList);
349
350     // for(i=0; i<task_num; i++){
351     // broker.bindCloudletToVm(cloudlet[i].getCloudletId(),vm_
352     // }
353
354
355
356
357
358     // Fourth step: Create one virtual machine
359     // vmList = createVms(brokerId);
360

```

```

361         // submit vm list to the broker
362         // broker.submitVmList(vmarray);
363
364         // Fifth step: Create one cloudlet
365         // broker.submitVmList(vmarray);
366         // cloudletList =
367             createCloudletList(brokerId);
368
369         // submit cloudlet list to the broker
370         // broker.submitCloudletList(cloudletList);
371
372         // Sixth step: Starts the simulation
373         double lastClock =
374             CloudSim.startSimulation();
375
376         // Final step: Print results when simulation
377         // is over
378         List<Cloudlet> newList =
379             broker.getCloudletReceivedList();
380
381         Log.println("Received " + newList.size() +
382             " cloudlets");
383
384         CloudSim.stopSimulation();
385
386         printCloudletList(datacenter,
387             vmList, newList);

```

```

386         Log.println();
387         Log.println(String.format("Total
                                     simulation time: %.2f sec", lastClock));
388         Log.println(String.format("Energy
                                     consumption: %.2f kWh",
                                     datacenter.getPower() / (3600 * 1000)));
389         Log.println(String.format("Number of VM
                                     migrations: %d",
                                     datacenter.getMigrationCount()));
390         //Log.println(String.format("Number of SLA
                                     violations: %d", sla.size()));
391         //Log.println(String.format("SLA violation
                                     percentage: %.2f%%", (double) sla.size()
                                     * 100 / numberOfAllocations));
392         //Log.println(String.format("Average SLA
                                     violation: %.2f%%", averageSla));
393         Log.println();
394
395     } catch (Exception e) {
396         e.printStackTrace();
397         Log.println("Unwanted errors happen");
398     }
399
400     Log.println("SingleThreshold finished!");
401 }
402
403 /**
404  * Creates the cloudlet list.
405  *
406  * @param brokerId the broker id

```



```

407      *
408      * @return the cloudlet list
409      */
410
411
412
413  /**
414   * Creates the vms.
415   *
416   * @param brokerId the broker id
417   *
418   * @return the list < vm>
419   */
420
421
422
423
424  private static Datacenter createDatacenter(String name,
425                                             Class<? extends Datacenter> datacenterClass)
426      throws Exception {
427      // Here are the steps needed to create a
428      // PowerDatacenter:
429      // 1. We need to create an object of HostList2 to
430      // store
431      // our machine
432      List<PowerHost> hostList = new
433      ArrayList<PowerHost>();
434
435      // Class<? extends Datacenter> datacenterClass;
436
437      double maxPower = 250; // 250W
438
439      double staticPowerPercent = 0.7; // 70%

```

```

434
435         int[] mips = { 5000, 5500 };
436         int [] ram = {4096, 4096}; // host memory (MB)
437         long storage = 10000; // host storage
438         int bw = 10000;
439         int[] HOST_MIPS = { 5000, 4500 };
440         int[] HOST_PES      = { 1, 1 };
441         for (int i = 0; i < hostsNumber; i++) {
442             int hostType = i % Constants.HOST_TYPES;
443
444             List<Pe> peList = new ArrayList<Pe>();
445             for (int j = 0; j <
                        Constants.HOST_PES[hostType]; j++) {
446                 peList.add(new Pe(j, new
                                PeProvisionerSimple(HOST_MIPS[hostType])));
447             }
448
449             hostList.add(new PowerHostUtilizationHistory(
450                 i,
451                 new
                                RamProvisionerSimple(ram[hostType]),
452                 new
                                BwProvisionerSimple( Constants.HOST_BW),
453                 Constants.HOST_STORAGE,
454                 peList,
455                 new
                                VmSchedulerTimeShared( peList),
456                 Constants.HOST_POWER[hostType])); //
                                This is our machine
457         }

```

```

458
459         // 5. Create a DatacenterCharacteristics object that
460         stores the
461         // properties of a Grid resource: architecture , OS,
462         list of
463         // Machines, allocation policy: time- or
464         space-shared, time zone
465         // and its price (G$/PowerPe time unit).
466         String arch = "x86"; // system architecture
467         String os = "Linux"; // operating system
468         String vmm = "Xen";
469         double time_zone = 10.0; // time zone this resource
470         located
471         double cost = 3.0; // the cost of using processing
472         in this resource
473         double costPerMem = 0.05; // the cost of using
474         memory in this resource
475         double costPerStorage = 0.001; // the cost of using
476         storage in this resource
477         double costPerBw = 0.0; // the cost of using bw in
478         this resource
479
480         String vmAllocationPolicyName="mad";
481             String vmSelectionPolicyName= "mmt";
482             String parameterName="2.5";
483             VmAllocationPolicy vmAllocationPolicy = null;
484         PowerVmSelectionPolicy vmSelectionPolicy = null;
485         if (!vmSelectionPolicyName.isEmpty()) {
486             vmSelectionPolicy =
487                 getVmSelectionPolicy(vmSelectionPolicyName);
488         }

```

```

479         double parameter = 0;
480         if (!parameterName.isEmpty()) {
481             parameter = Double.valueOf(parameterName);
482         }
483
484
485         if (vmAllocationPolicyName.equals("iqr")) {
486             PowerVmAllocationPolicyMigrationAbstract
487                 fallbackVmSelectionPolicy = new
488                     PowerVmAllocationPolicyMigrationStaticThreshold(
489                         hostList ,
490                         vmSelectionPolicy ,
491                         0.7);
492             vmAllocationPolicy = new
493                 PowerVmAllocationPolicyMigrationInterQuartileRange(
494                     hostList ,
495                     vmSelectionPolicy ,
496                     parameter ,
497                     fallbackVmSelectionPolicy);
498         } else if (vmAllocationPolicyName.equals("mad")) {
499             PowerVmAllocationPolicyMigrationAbstract
500                 fallbackVmSelectionPolicy = new
501                     PowerVmAllocationPolicyMigrationStaticThreshold(
502                         hostList ,
503                         vmSelectionPolicy ,
504                         0.7);
505             vmAllocationPolicy = new
506                 PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation
507                     hostList ,
508                     vmSelectionPolicy ,

```

```

503             parameter ,
504             fallbackVmSelectionPolicy);
505     } else if (vmAllocationPolicyName.equals("lr")) {
506         PowerVmAllocationPolicyMigrationAbstract
507             fallbackVmSelectionPolicy = new
508             PowerVmAllocationPolicyMigrationStaticThreshold(
509                 hostList ,
510                 vmSelectionPolicy ,
511                 0.7);
512     vmAllocationPolicy = new
513         PowerVmAllocationPolicyMigrationLocalRegression(
514             hostList ,
515             vmSelectionPolicy ,
516             parameter ,
517             Constants.SCHEDULING_INTERVAL,
518             fallbackVmSelectionPolicy);
519     } else if (vmAllocationPolicyName.equals("lrr")) {
520         PowerVmAllocationPolicyMigrationAbstract
521             fallbackVmSelectionPolicy = new
522             PowerVmAllocationPolicyMigrationStaticThreshold(
523                 hostList ,
524                 vmSelectionPolicy ,
525                 0.7);
526     vmAllocationPolicy = new
527         PowerVmAllocationPolicyMigrationLocalRegressionRobust(
528             hostList ,
529             vmSelectionPolicy ,
530             parameter ,
531             Constants.SCHEDULING_INTERVAL,
532             fallbackVmSelectionPolicy);

```

```

527         } else if (vmAllocationPolicyName.equals("thr")) {
528             vmAllocationPolicy = new
529                 PowerVmAllocationPolicyMigrationStaticThreshold(
530                     hostList ,
531                     vmSelectionPolicy ,
532                     parameter);
533         } else if (vmAllocationPolicyName.equals("dvfs")) {
534             vmAllocationPolicy = new
535                 PowerVmAllocationPolicySimple(hostList);
536         } else {
537             System.out.println("Unknown VM allocation
538                 policy: " + vmAllocationPolicyName);
539             System.exit(0);
540         }
541
542     DatacenterCharacteristics characteristics = new
543         DatacenterCharacteristics(
544             arch , os , vmm, hostList , time_zone ,
545             cost , costPerMem , costPerStorage ,
546             costPerBw);
547
548     // 6. Finally , we need to create a PowerDatacenter
549     // object.
550
551     Datacenter datacenter = null;
552
553     try {
554         datacenter = datacenterClass.getConstructor(
555             String.class ,
556             DatacenterCharacteristics.class ,
557             VmAllocationPolicy.class ,

```

```

550         List.class ,
551         Double.TYPE).newInstance(
552         name ,
553         characteristics ,
554         vmAllocationPolicy ,
555         new LinkedList<Storage>() ,
556         Constants.SCHEDULING_INTERVAL);
557     } catch (Exception e) {
558         e.printStackTrace();
559     }
560
561     return datacenter;
562 }
563 protected static PowerVmSelectionPolicy getVmSelectionPolicy( String
    vmSelectionPolicyName) {
564     PowerVmSelectionPolicy vmSelectionPolicy = null;
565
566     if (vmSelectionPolicyName.equals("mc")) {
567         vmSelectionPolicy = new
568             PowerVmSelectionPolicyMaximumCorrelation(
569                 new
570                     PowerVmSelectionPolicyMinimumMigrationTime()
571             );
572     } else if (vmSelectionPolicyName.equals("mmt")) {
573         vmSelectionPolicy = new
574             PowerVmSelectionPolicyMinimumMigrationTime();
575     } else if (vmSelectionPolicyName.equals("mu")) {
576         vmSelectionPolicy = new
577             PowerVmSelectionPolicyMinimumUtilization();
578     } else if (vmSelectionPolicyName.equals("rs")) {
579         vmSelectionPolicy = new
580             PowerVmSelectionPolicyRoundRobin();
581     }
582
583     return vmSelectionPolicy;
584 }

```

```

574         vmSelectionPolicy = new
                    PowerVmSelectionPolicyRandomSelection();
575     } else {
576         System.out.println("Unknown VM selection
                    policy: " + vmSelectionPolicyName);
577         System.exit(0);
578     }
579     return vmSelectionPolicy;
580 }
581 // We strongly encourage users to develop their own broker
    policies , to
582 // submit vms and cloudlets according
583 // to the specific rules of the simulated scenario
584 /**
585  * Creates the broker.
586  *
587  * @return the datacenter broker
588  */
589 private static DatacenterBroker createBroker() {
590     DatacenterBroker broker = null;
591     try {
592         broker = new DatacenterBroker("Broker");
593     } catch (Exception e) {
594         e.printStackTrace();
595         return null;
596     }
597     return broker;
598 }
599
600 /**

```



```

601      * Prints the Cloudlet objects.
602      *
603      * @param list list of Cloudlets
604      */
605      private static void printCloudletList(PowerDatacenter
        datacenter ,
606
        List<Vm> vms, List<Cloudlet> list) {
607          int size = list.size();
608          List<Host> hosts = datacenter.getHostList();
609
610          int numberOfHosts = hosts.size();
611          int numberOfVms = vms.size();
612          Map<String , Double> slaMetrics = getSlaMetrics(vms);
613          double slaOverall = slaMetrics.get("overall");
614          double slaAverage = slaMetrics.get("average");
615          double slaDegradationDueToMigration =
        slaMetrics.get("underallocated_migration");
616          double slaTimePerActiveHost =
        getSlaTimePerActiveHost(hosts);
617
618          double sla = slaTimePerActiveHost *
        slaDegradationDueToMigration;
619          List<Double> timeBeforeHostShutdown =
        getTimesBeforeHostShutdown(hosts);
620
621          int numberOfHostShutdowns =
        timeBeforeHostShutdown.size();
622
623          Cloudlet cloudlet;
624

```

```

625         String indent = "\t";
626         Log.println();
627
628         Log.println();
629         Log.println(String.format("%d",
        numberOfHosts));
630         Log.println(String.format("%d",
        numberOfVms));
631         Log.println(String.format("SLA: %.5f%%",
        sla * 100));
632         Log.println(String.format("SLA time per
        active host: %.2f%%",
        slaTimePerActiveHost * 100));
633         Log.println(String.format("SLA perf
        degradation due to migration: %.2f%%",
634                                 slaDegradationDueToMigration
        * 100));
635         Log.println(String.format("Overall SLA
        violation: %.2f%%", slaOverall * 100));
636         Log.println(String.format("Average SLA
        violation: %.2f%%", slaAverage * 100));
637         Log.println(String.format("Number of host
        shutdowns: %d", numberOfHostShutdowns));
638
639     }
640     protected static Map<String, Double> getSlaMetrics(List<Vm>
        vms) {
641         Map<String, Double> metrics = new HashMap<String,
        Double>();
642         List<Double> slaViolation = new LinkedList<Double>();

```

```

643         double totalAllocated = 0;
644         double totalRequested = 0;
645         double totalUnderAllocatedDueToMigration = 0;
646
647         for (Vm vm : vms) {
648             double vmTotalAllocated = 0;
649             double vmTotalRequested = 0;
650             double vmUnderAllocatedDueToMigration = 0;
651             double previousTime = -1;
652             double previousAllocated = 0;
653             double previousRequested = 0;
654             boolean previousIsInMigration = false;
655
656             for (VmStateHistoryEntry entry :
657                 vm.getStateHistory()) {
658                 if (previousTime != -1) {
659                     double timeDiff =
660                         entry.getTime() -
661                         previousTime;
662                     vmTotalAllocated +=
663                         previousAllocated *
664                         timeDiff;
665                     vmTotalRequested +=
666                         previousRequested *
667                         timeDiff;
668
669                     if (previousAllocated <
670                         previousRequested) {
671                         slaViolation.add((previousRequested

```

—

```

previousAllocated)
/
previousRequested);
664         if
            (previousIsInMigration)
            {
665                 vmUnderAllocatedDueToMigration
                    +=
                    (previousRequested
                    -
                    previousAllocated)
666                                     *
timeDif
667             }
668     }
669 }
670
671     previousAllocated =
        entry.getAllocatedMips();
672     previousRequested =
        entry.getRequestMips();
673     previousTime = entry.getTime();
674     previousIsInMigration =
        entry.isInMigration();
675 }
676
677     totalAllocated += vmTotalAllocated;
678     totalRequested += vmTotalRequested;
679     totalUnderAllocatedDueToMigration +=
        vmUnderAllocatedDueToMigration;

```

```

680         }
681
682         metrics.put("overall", (totalRequested -
            totalAllocated) / totalRequested);
683         if (slaViolation.isEmpty()) {
684             metrics.put("average", 0.);
685         } else {
686             metrics.put("average",
                MathUtil.mean(slaViolation));
687         }
688         metrics.put("underallocated_migration",
            totalUnderAllocatedDueToMigration /
            totalRequested);
689         // metrics.put("sla_time_per_vm_with_migration",
            slaViolationTimePerVmWithMigration /
690         // totalTime);
691         // metrics.put("sla_time_per_vm_without_migration",
            slaViolationTimePerVmWithoutMigration /
692         // totalTime);
693
694         return metrics;
695     }
696     protected static double getSlaTimePerActiveHost(List<Host>
        hosts) {
697         double slaViolationTimePerHost = 0;
698         double totalTime = 0;
699
700         for (Host _host : hosts) {
701             HostDynamicWorkload host =
                (HostDynamicWorkload) _host;

```

```

702         double previousTime = -1;
703         double previousAllocated = 0;
704         double previousRequested = 0;
705         boolean previousIsActive = true;
706
707         for (HostStateHistoryEntry entry :
              host.getStateHistory()) {
708             if (previousTime != -1 &&
                previousIsActive) {
709                 double timeDiff =
                    entry.getTime() -
                    previousTime;
710                 totalTime += timeDiff;
711                 if (previousAllocated <
                    previousRequested) {
712                     slaViolationTimePerHost
                        += timeDiff;
713                 }
714             }
715
716             previousAllocated =
                entry.getAllocatedMips();
717             previousRequested =
                entry.getRequestMips();
718             previousTime = entry.getTime();
719             previousIsActive = entry.isActive();
720         }
721     }
722
723     return slaViolationTimePerHost / totalTime;

```

```

724     }
725     public static List<Double>
        getTimesBeforeHostShutdown(List<Host> hosts) {
726         List<Double> timeBeforeShutdown = new
            LinkedList<Double>();
727         for (Host host : hosts) {
728             boolean previousIsActive = true;
729             double lastTimeSwitchedOn = 0;
730             for (HostStateHistoryEntry entry :
                ((HostDynamicWorkload)
                    host).getStateHistory()) {
731                 if (previousIsActive == true &&
                    entry.isActive() == false) {
732                     timeBeforeShutdown.add(entry.getTime()
                        - lastTimeSwitchedOn);
733                 }
734                 if (previousIsActive == false &&
                    entry.isActive() == true) {
735                     lastTimeSwitchedOn =
                        entry.getTime();
736                 }
737                 previousIsActive = entry.isActive();
738             }
739         }
740         return timeBeforeShutdown;
741     }
742
743
744
745

```


Abstract:

Today, with the advent of information technology and the rise of applications, there is no doubt a need for an integrated calculation for users. Therefore, it is necessary to use technology such as a computer that performs their processing according to the needs of the users and displays the results to them. At the moment, there are a variety of cloud computing challenges. Effective use, allocation and management of resources to improve the quality of service and energy efficiency is one of the major challenges in cloud systems. The thesis focuses on choosing the proper destination for hosting virtual machines and controlling the migration of migrating virtual machines. Our goal is to automate the assignment of virtual machines to physical hosts so as to prevent the overload of physical hosts and extra migration. Experimental results compared to the base method show that with proper allocation and control in migration, we can improve the quality of service, while avoiding increasing energy consumption.

Keywords: Cloud computing, Allocation, Overload, Control of migration, Quality of Service, Energy Consumption.



**Islamic Azad University
Sirjan Branch**

Faculty of Engineering Department of Computer Engineering

M .Sc Thesis

A new method to assign virtual machines to avoid overloading the physical host to improve the quality of service in the cloud data center

Thesis Advisor :

Dr Mohammad Sadegh Hajmohammadi

By:

Molaei Saeb

Summer 2017