الگوریتم شکارچیان دریایی آگاه از انرژی برای زمان‌بندی وظایف در برنامه‌های محاسباتی مه مبتنی بر اینترنت اشیا

هانیه ملائی[1]

[1] دانشجوی کارشناسی ارشد، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف تهران، ایران
Haniye.mollaei32@sharif.edu

چکیده

در عصر حاضر که شاهد پیشرفت‌های چشمگیر در فناوری‌های اینترنت اشیاء و محاسبات مه هستیم، یکی از چالش‌های کلیدی، بهینه‌سازی و زمانبندی وظایف در این سیستم‌ها است. زمانبندی مؤثر وظایف در محاسبات مه، نه تنها بر کیفیت خدمات[1] تأثیر می‌گذارد، بلکه باعث بهبود کارایی سیستم، کاهش مصرف انرژی و کاهش تأثیرات زیست‌محیطی می‌شود. در این راستا، مقاله اصلی یک رویکرد نوآورانه با به‌کارگیری الگوریتم فرا ابتکاری[2] شکارچیان دریایی[3] ارائه می‌دهد. در این مطالعه، علاوه بر بررسی نظری این الگوریتم آگاه از انرژی و نسخه‌های بهبود یافته‌ی آن، با پیاده‌سازی عملی در محیط برنامه‌نویسی جاوا، به ارزیابی کارایی آن‌ها در محیط‌های واقعی می‌پردازیم. این پیاده‌سازی نشان‌دهنده تأثیر قابل توجه الگوریتم فرا ابتکاری شکارچیان دریایی، به‌خصوص نسخه بهبود یافته‌ی آن، در بهبود کارایی سیستم‌های محاسبات مه و کاهش مصرف انرژی است. استفاده از این الگوریتم‌ها نه تنها در کاهش زمان پاسخ‌دهی و تأثیرات زیست‌محیطی نقش مهمی دارد، بلکه می‌تواند به افزایش کارایی کلی سیستم‌های محاسبات مه مبتنی بر اینترنت اشیاء کمک کند.

کلمات کلیدی

الگوریتم شکارچیان دریایی، زمان‌بندی، اینترنت اشیا، محاسبات مه، الگوریتم فراابتکاری، کیفیت خدمات

۱- مقدمه

در عصر حاضر، اینترنت اشیا[4] به عنوان یک شبکه‌ی گسترده از دستگاه‌های متصل به یکدیگر، نقش حیاتی در زندگی روزمره و صنایع مختلف ایفا می‌کند. اینترنت اشیا با اتصال میلیاردها دستگاه هوشمند در سراسر جهان، موجب تولید حجم عظیمی از داده‌ها می‌شود. این داده‌ها نه تنها نیازمند ذخیره‌سازی هستند، بلکه باید تحلیل و پردازش شوند تا اطلاعات سودمندی را فراهم آورند. در این راستا، محاسبات ابری به عنوان یک راه‌حل مهم برای ذخیره‌سازی و پردازش این داده‌ها مطرح می‌شود. با این حال، محاسبات ابری با محدودیت‌هایی از جمله تاخیر در پاسخ‌دهی و نیاز به پهنای باند بالا روبرو است.

برای رفع این چالش‌ها، مفهوم محاسبات مه[5] مطرح شده است که به عنوان یک لایه واسط بین دستگاه‌های اینترنت اشیا و مراکز داده ابری عمل می‌کند. محاسبات مه با فراهم کردن توان پردازشی و ذخیره‌سازی در نزدیکی دستگاه‌های اینترنت اشیا، تاخیر را کاهش داده و پاسخ‌گویی سریع‌تری را ارائه می‌دهد. این مفهوم در زمینه‌های متنوعی مانند بهداشت هوشمند، شهرهای هوشمند و مدیریت ترافیک کاربرد دارد.

در این زمینه، چالش اساسی مربوط به تخصیص و برنامه‌ریزی منابع و تسک‌ها در محیط محاسبات مه است. این چالش به دلیل تنوع و پیچیدگی تسک‌ها و محدودیت منابع در محاسبات مه، اهمیت زیادی دارد. برای مدیریت بهینه این منابع، الگوریتم‌های مختلفی پیشنهاد شده‌اند که از جمله آن‌ها می‌توان به الگوریتم‌های فراابتکاری اشاره کرد. این الگوریتم‌ها با استفاده از روش‌های اکتشافی و استثماری به دنبال یافتن راه‌حل‌های مطلوب در فضای جستجوی گسترده هستند.

در این پژوهش، تمرکز ما بر روی الگوریتم MPA[6] است. MPA که به عنوان یک الگوریتم فراابتکاری برای حل مسائل بهینه‌سازی پیوسته مطرح شده، در محاسبات مه برای برنامه‌ریزی وظایف به کار

[1] QoS(Quality Of Services)
[2] Metaheuristic
[3] Marine Predators Algorithm
[4] IoT
[5] Fog Computing
[6] Marine Predator Algorithm

گرفته می‌شود. این الگوریتم با توانایی بالا در اکتشاف و استثمار، نتایج قابل توجهی را در حل چالش‌های مرتبط با وظایف محاسبات مه ارائه می‌دهد. به منظور بهبود عملکرد MPA، نسخه تعدیل شده‌ای به نام MMPA[1] پیشنهاد شده است که با تغییراتی در روش‌های به‌روزرسانی و انتخاب راه‌حل‌ها، به دنبال بهینه‌سازی بیشتر است.

این پژوهش به بررسی و تحلیل عمیق الگوریتم‌های MPA، MMPA و IMMPA[2] می‌پردازد تا اثربخشی آن‌ها در مدیریت بهینه تسک‌ها در محاسبات مه را مورد ارزیابی قرار دهد. هدف اصلی، کاهش زمان پاسخ، بهبود کارایی و کاهش مصرف انرژی در محیط‌های محاسبات مه است. این پژوهش با مقایسه عملکرد این الگوریتم‌ها با سایر روش‌های متداول، سعی در ارائه راهکارهای نوآورانه برای بهبود کیفیت خدمات (QoS) در این حوزه دارد.

در بخش‌های بعدی این مقاله، به تفصیل به توضیح الگوریتم‌های MPA، MMPA و IMMPA و نحوه اقتباس و بهبود آن‌ها برای استفاده در محیط‌های محاسبات مه می‌پردازیم. همچنین به بررسی کارهای مرتبط در زمینه برنامه‌ریزی تسک‌ها در محاسبات مه پرداخته می‌شود که شامل معرفی و تحلیل مختصری از رویکردها و الگوریتم‌هایی مانند الگوریتم سینوس و کسینوس، الگوریتم بهینه‌سازی وال و الگوریتم بهینه‌سازی ذرات است. تمرکز اصلی در این موارد نیز بر روی نحوه بهینه‌سازی این الگوریتم‌ها برای مواجهه با چالش‌های خاص محاسبات مه، از جمله تعادل بار، کاهش زمان پاسخ، کاهش انرژی و بهبود کیفیت خدمات است.

این مقایسه بر اساس معیارهای مختلف عملکردی مانند زمان اتمام کلی[3]، مصرف انرژی[4]، زمان اجرای کلی[5]، نرخ انتشار دی اکسید کربن [6]و تابع تناسب[7] انجام می‌شود. هدف از این مقایسه، ارزیابی کارایی و اثربخشی الگوریتم‌های پیشنهادی در مقابله با چالش‌های محاسبات مه و ارائه یک راه‌حل جامع برای بهبود کیفیت خدمات در این حوزه است. در نهایت، با توجه به نتایج به دست آمده، پیشنهاداتی برای کارهای آتی در این زمینه ارائه می‌شود تا گامی به سوی بهبود بیشتر عملکرد سیستم‌های مبتنی بر محاسبات مه برداشته شود.

۱-۱- معیارهای مورد ارزیابی

در مقاله‌ای که مورد بررسی قرار گرفته است، چندین پارامتر کلیدی برای ارزیابی عملکرد للگوریتم‌های MPA، MMPA، و IMMPA مورد استفاده قرار گرفته‌اند. این پارامترها شامل مصرف انرژی، زمان اتمام کل، زمان اجرای کل و نرخ انتشار دی اکسید کربن هستند. برای هر یک از این پارامترها، نحوه محاسبه و اهمیت آنها در ارزیابی عملکرد سیستم‌های محاسبات مه توضیح داده خواهد شد.

مصرف انرژی به مقدار انرژی مورد نیاز برای اجرای کل تسک‌ها در یک محیط محاسبات مه اشاره دارد. این معیار با توجه به میزان مصرف انرژی هر واحد محاسباتی (مانند ماشین‌های مجازی) و زمان اجرای تسک‌ها بر روی آن‌ها محاسبه می‌شود. فرمول مربوطه بر اساس توان مصرفی و زمان فعالیت دستگاه‌ها تعیین می‌شود که شامل انرژی مصرفی در حالت فعال و حالت آماده‌به‌کار است. کاهش مصرف انرژی به معنای افزایش بهره‌وری انرژی و کاهش هزینه‌های عملیاتی است.

$$E_{\text{total}} = \sum_{i=1}^{N} E(VM_i) = \sum_{i=1}^{N} \left(E_{\text{active},i} + E_{idle,i}\right) \quad (1)$$

زمان اتمام کلی، مدت زمان لازم برای اتمام تمام وظایف در سیستم است. این معیار نشان‌دهنده کارایی کلی سیستم در پردازش وظایف است و تأثیر مستقیمی بر رضایت کاربران دارد. این معیار با بررسی زمان شروع و پایان هر وظیفه و یافتن بیشترین زمان پایانی بین تمام وظایف محاسبه می‌شود.

$$MK = \max\left(Et_j\right) \text{ for } j = 1,2,\dots,n \quad (2)$$

زمان اجرای کل، مجموع زمان‌های انتظار و اجرای تمام وظایف در سیستم است. این معیار به تحلیل میزان تأخیر در پردازش وظایف کمک می‌کند و برای ارزیابی کارایی توزیع بار، سرعت پردازش و مدیریت منابع در محاسبات مه مهم است. فرمول مربوطه شامل مجموع زمان‌های انتظار و اجرا برای هر وظیفه است.

$$\text{Flow\_Time} = \sum_{i=1}^{n} f_i \quad (3)$$

نرخ انتشار دی‌اکسید کربن به عنوان یک معیار زیست‌محیطی، اهمیت فزاینده‌ای یافته است. این معیار بر اساس مقدار دی‌اکسید کربن تولید شده توسط منابع انرژی مورد استفاده برای اجرای وظایف محاسبه می‌شود و با توجه به نوع منابع انرژی و میزان مصرف انرژی تعیین می‌گردد.

$$\text{CO2\_Emission} = \sum_{k=1}^{4} (\text{Total\_Energy} \times \text{sh}_k \times \text{emission\_factor}_k \times \text{ratio}) \quad (4)$$

---

[1] Modified Marine Predator Algorithm
[2] Improved Marine Predator Algorithm
[3] Makespan
[4] Energy Consumption
[5] Flow time
[6] Carbon dioxide emission rate
[7] Fitness function

تابع تناسب یکی از اجزای کلیدی در الگوریتم‌های فراابتکاری مانند MPA، MMPA و IMMPA است که به ارزیابی و رتبه‌بندی راه‌حل‌های مختلف بر اساس میزان مطلوبیت آن‌ها کمک می‌کند. این تابع با در نظر گرفتن معیارهای مختلفی مانند مصرف انرژی، زمان اتمام کلی و زمان جریان کلی، یک مقدار عددی را به هر راه‌حل اختصاص می‌دهد که نشان‌دهنده کیفیت آن راه‌حل در حل مسئله است.

در زمینه محاسبات مه، تابع تناسب معمولاً به گونه‌ای تعریف می‌شود که همزمان به دنبال کمینه‌سازی مصرف انرژی و زمان پردازش باشد. برای مثال، می‌توان تابع تناسبی را تعریف کرد که ترکیبی از مصرف انرژی و زمان اتمام کلی باشد. فرمول مربوطه می‌تواند به صورت زیر باشد:

$$\text{Fitness} = \alpha \times \text{Total\_Energy} + (1-\alpha) \times MK \quad (5)$$

که در آن α ضریبی است که وزن مصرف انرژی و زمان اتمام کلی را در تابع تناسب تعیین می‌کنند. این ضریب باید به گونه‌ای انتخاب شوند که تعادل مناسبی بین دو معیار ایجاد کنند. اما در این مطالعه به دلیل تمرکز بیشتر بر کاهش مصرف انرژی و در تبع آن تولید دی‌اکسید کربن وزن ۰.۸ به انرژی و ۰.۲ به طول زمان اجرای کل نسبت داده می‌شود.

۲-۱- الگوریتم پایه MPA

الگوریتم شکارچیان دریایی یک تکنیک بهینه‌سازی فراابتکاری است که از رفتار شکار شکارچیان دریایی مانند کوسه‌ها، دلفین‌ها و نهنگ‌ها الهام گرفته شده است. می‌توان از آن برای حل مسائل مختلف بهینه‌سازی، از جمله زمان‌بندی وظایف در محاسبات مه استفاده کرد.

این للگوریتم از دو نوع راه‌پیمایی تصادفی برای کاوش و بهره‌برداری از فضای جستجو استفاده می‌کند: لووی [۱] و براونی [۲]. همچنین با نسبت‌های سرعت مختلف بین شکارگر و شکار تطابق پیدا می‌کند و تأثیرات محیطی مانند تشکیل گرداب و دستگاه‌های جلب ماهی [۳] را مد نظر قرار می‌دهد.

در زیر چند نمونه از نحوه عملکرد MPA برای مسائل بهینه‌سازی مختلف آورده شده است:

در [1] MPA برای حل مسائل بهینه‌سازی محدود [۴] به‌کار رفته است، که مسائل بهینه‌سازی هستند که یک یا چند محدودیت روی منطق ممکن ناحیه فضای جستجو دارند. در این مطالعه MPA با یک تکنیک پردازش محدودیت مبتنی بر قوانین امکان‌پذیری و انتخاب تورنامنت بهبود یافته است. نتایج نشان داد که MPA در مقایسه با سایر الگوریتم‌ها مانند تکامل تفاضلی، جستجوی هارمونی و جستجوی کوکو در چندین COPs مقیاس‌بندی بهتری داشته است.

در [2] MPA برای حل مسائل بهینه‌سازی چند هدفه [۵] توسعه داده شد، که مسائل بهینه‌سازی هستند که بیش از یک تابع هدف دارند که باید به طور همزمان بهینه شوند. چهار نوع MPA برای MOPها پیشنهاد شد، یعنی الگوریتم شکارچیان دریایی چند هدفه [۶]، مرتب‌سازی غیر غالب [7] ، فاصله ازدحام [۸]، و تجزیه [۹]. عملکرد این گونه‌ها با سایر الگوریتم‌های چند هدفه مانند الگوریتم ژنتیک مرتب‌سازی غیرمسلط [۱۰] ، بهینه‌سازی ازدحام ذرات چند هدفه [۱۱] و الگوریتم تکاملی چند هدفه مبتنی بر تجزیه [۱۲] مقایسه شد.

این الگوریتم شامل چند مرحله است که در زیر به صورت کلی آن را مرور می‌کنیم.

مرحله‌ی اولیه:

همانند بیشتر الگوریتم‌های فراابتکاری،MPA با پخش کردن راه‌حل‌های خود در فضای جستجوی مسئله آغاز می‌شود. این کار با استفاده از فرمول زیر انجام می‌شود:

$$\vec{X} = \vec{X}_L + \text{rand} * (\vec{X}_U - \vec{X}_L) \quad (6)$$

که در آن rand یک عدد تصادفی در بازه ۰ تا ۱ است و $\vec{X}_L$ و $\vec{X}_U$ حداقل و حداکثر مرز فضای جستجو برای یک مسئله بهینه‌سازی هستند.

ساخت ماتریس نخبگان و شکار [۱۳]:

بهترین شکارچی به عنوان بهترین عنصر برای فرآیند جستجو محسوب می‌شود و برای ساخت ماتریس نخبگان به کار می‌رود. این ماتریس با هر تکرار بر اساس بهترین شکارچی یافت شده ساخته می‌شود و هر تکرار با به دست آوردن یک شکارچی بهتر به‌روزرسانی می‌شود.

$$E = \begin{bmatrix} X_{1,1}^I & X_{1,2}^I & \dots & X_{1,d}^I \\ X_{2,1}^I & X_{2,2}^I & \dots & X_{2,d}^I \\ \vdots & \vdots & \vdots & \vdots \\ X_{n,1}^I & X_{n,2}^I & \dots & X_{n,d}^I \end{bmatrix}$$

ماتریس شکار نیز به همین ترتیب ساخته می‌شود و فرآیند بهینه‌سازی بر اساس این دو معیار پایه‌ریزی شده است.

---

[۱] Lévy
[۲] Brownian
[۳] FADs
[۴] COPs
[۵] MOPs
[۶] MMPA
[7] NS-MMPA
[۸] CD-MMPA
[۹] D-MMPA
[۱۰] NSGA-II
[۱۱] MOPSO
[۱۲] MOEA
[۱۳] Elite and Prey Matrix

$$py = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,d} \\ X_{2,1} & X_{2,2} & \dots & X_{2,d} \\ \vdots & \vdots & : & \vdots \\ X_{n,1} & .X_{n,2} & \dots & X_{n,d} \end{bmatrix}$$

n نشانگر تعداد تکرار و d نشانگر طول راه حل است. در هر تکرار بهترین شکارچی تا آن تکرار را در این ماتریس ذخیره می‌کنیم.

به صورت کلی در مقاله، ماتریس نخبگان نمایانگر بهترین راه‌حل یافت شده تا به حال است که در سطرهای ماتریس تکرار شده تا به اندازه جمعیت برسد. این ماتریس به عنوان مرجعی برای حرکت شکارچیان در فرایند بهینه‌سازی به کار می‌رود. از طرف دیگر، ماتریس شکار به طور پویا به‌روز رسانی می‌شود و نمایانگر موقعیت‌های شکار در فضای جستجو است. شکارچیان به سمت شکار حرکت می‌کنند و موقعیت خود را بر اساس فاصله بین ماتریس نخبگان (بهترین راه‌حل) و ماتریس شکار (راه‌حل‌های فعلی) تنظیم می‌کنند، که این امر به کاوش و بهره‌برداری در فضای جستجو کمک می‌کند.

برای مثال، اگر ماتریس نخبگان نشان‌دهنده موقعیت‌هایی با کمترین میزان مصرف انرژی در یک سیستم محاسبات ابری باشد، ماتریس شکار می‌تواند موقعیت‌های فعلی راه‌حل‌ها را با توجه به میزان مصرف انرژی و سایر پارامترهای مهم نشان دهد. شکارچیان (الگوریتم‌های بهینه‌سازی) سعی می‌کنند با حرکت به سمت موقعیت‌های بهتر در ماتریس شکار، راه‌حل‌های مطلوب‌تری را بیابند و به ماتریس نخبگان نزدیک شوند.

فرآیند بهینه‌سازی:

فرآیند بهینه‌سازی در MPA شامل سه مرحله اصلی است:

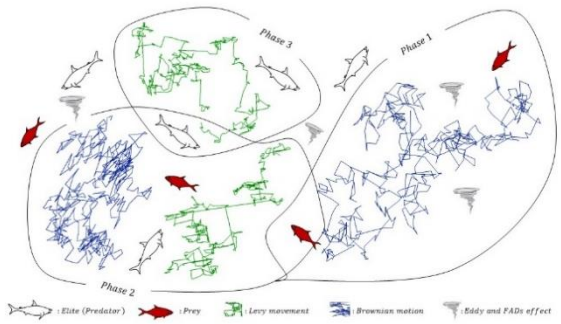
(۱) مرحله کاوش: در این مرحله، شکارچیان سریع‌تر حرکت می‌کنند تا شکار خود را پیدا کنند. در این مرحله راه حل‌ها با گام براونی بروزرسانی می‌شوند.

(۲) مرحله میانی: این مرحله بین کاوش و استفاده تعادل برقرار می‌کند. جمعیت به دو نیم تقسیم می‌شود: نیمه اول با گام براونی و نیمه دوم با گام لویی به‌روزرسانی می‌شوند.

(۳) مرحله نهایی: در این مرحله، تمام جمعیت تنها با استفاده از گام لویی به‌روزرسانی می‌شوند.

۱-۳- الگوریتم MMPA

MMPA نسخه اصلاح‌شده الگوریتم MPA است که قابلیت بهره‌برداری آن را افزایش می‌دهد. برخلاف MPA که ممکن است برای به‌روزرسانی‌ها از آخرین موقعیت بهترین راهکار استفاده کند،



شکل ۱ – شمایی از اجرای الگوریتم شکارچیان دریایی

MMPA از آخرین موقعیت‌های به‌روزرسانی‌شده استفاده می‌کند و جستجوی مخرب‌تری به سمت راهکار بهینه را ترویج می‌کند.

این رویکرد همیشه با استفاده از آخرین موقعیت شکارچیان برای محاسبات دوره‌های بعدی بررسی می‌شود، بدون توجه به اینکه آیا بهتر از بهترین قبلی هستند یا نه. این تغییر به این معناست که احتمالاً شکارچیان به سرعت‌تر به سمت راهکارهای بهتری حرکت می‌کنند، با ریسک اینکه گاهی اوقات از موقعیت‌های بهترین شناخته‌شده دور شوند.

۱-۴- الگوریتم IMMPA

IMMPA توانایی جلوگیری از افتادن در نقاط محلی را بهبود می‌دهد و بهینه‌سازی شده‌تر از MMPA است. این الگوریتم با دوره‌هایی از بازآغازی بخشی از جمعیت و هدایت برخی افراد بهترین راهکارها، تنوع را تقویت می‌کند و جستجو را در اطراف راهکارهای پرامیده تشدید می‌کند. این رویکرد به تعادل بهبود یافته‌ای بین کاوش (بررسی مناطق جدید بالقوه) و بهره‌برداری (استفاده از مناطق معتبر شناخته‌شده) دست یافته و بهبود فرآیند جستجو و کیفیت راهکارها را هدف قرار می‌دهد.

۱-۵- الگوریتم سینوس کسینوس

بر اساس توابع ریاضی سینوس و کوسین، به هدایت عوامل جستجو به سمت بهترین راهکار از طریق یک سری تبدیلات ریاضی می‌پردازد.

از توابع سینوس و کوسینوس برای به‌روزرسانی موقعیت عوامل جستجو استفاده می‌کند، تا توازن بین کاوش (جستجوی مناطق جدید) و بهره‌برداری (افراز مناطق خوب شناخته‌شده) ارائه دهد. [3]

۶-۱- الگوریتم وال

شبیه‌سازی رفتار اجتماعی و مکانیزم شکار نهنگ‌های کوسه‌ای است. از روش تغذیه حباب شبکه در جستجوی خود استفاده می‌کند، با ترکیب رفتار حلقه زدن انقباضی و به‌روزرسانی موقعیت مارپیچی.

این الگوریتم رفتار نهنگ در حلقه زدن شکار و ایجاد حباب‌های شبکه را به طور موثر شبیه‌سازی می‌کند، تا به خوبی بین مراحل کاوش و بهره‌برداری توازن برقرار کند. [4]

۷-۱- الگوریتم بهینه‌سازی ذرات

مستلهم از رفتار اجتماعی قطیفه‌های پرنده یا گروه‌بندی ماهی‌ها است. تمرکز روی ذرات (عوامل) حرکت در فضای جستجو برای یافتن بهترین راهکار دارد.

هر ذره بر اساس تجربه خود و تجربه ذرات همسایه، پرواز خود را تنظیم می‌کند، تا اطلاعات مشترکی را به اشتراک بگذارد و گروه را به سمت بهترین راهکارها هدایت کند.

۸-۱- مجموعه داده‌های مورد آزمایش

در مقاله‌ی مرجع تمام الگوریتم‌های مورد مقایسه و سه نسخه پیشنهادی با استفاده از برنامه‌نویسی جاوا پیاده‌سازی شده و بر روی یک دستگاه با پردازنده Intel® Core™ i3-2330M CPU @ 2.20 GHz، ۱GB حافظه رم و مجهز به سیستم‌عامل Windows 7 Ultimate اجرا شده‌اند. برنامه‌ی نوشته شده در بازپیاده‌سازی ما روی Intel® Core™ i9-13900H CPU @ 2600 MHz، 16 GB حافظه رم و مجهز به سیستم‌عامل Windows 11 Ultimate اجرا شده است.

داده‌هایی که مطالعه‌ی ما بر روی آنها آزمایش خواهند شد به دو دسته تقسیم می‌شوند. دسته اول شامل ۱۲ وظیفه گوناگون با طول‌های ۱۰۰، ۲۰۰، ۳۰۰، ۴۰۰، ۵۰۰، ۶۰۰، ۷۰۰، ۸۰۰، ۹۰۰، ۱۰۰۰، ۱۲۰۰ و ۱۵۰۰ است. تمامی ۱۲ وظیفه بر روی تعداد ثابتی از ماشین‌های مجازی که ۶۰ عدد است، اجرا خواهند شد. در رابطه با بار کاری هر وظیفه، بارهای کاری گوناگون و به‌صورت تصادفی در محدوده ۱۰۰۰ تا ۱۰۰۰۰ تولید می‌شوند. در مورد سرعت پردازش مربوط به هر یک از ۶۰ ماشین مجازی، نیمه اول به ۲۰۰۰ MIPS و نیمه دوم به ۴۰۰۰ MIPS تنظیم شده‌اند.

مجموعه داده‌های دوم شامل یک وظیفه با طول ثابت ۶۰۰ است که این وظیفه بر روی دوازده ماشین مجازی با طول‌های ۱۰، ۲۰، ۳۰، ۴۰، ۵۰، ۶۰، ۷۰، ۸۰، ۹۰، ۱۰۰، ۱۲۰ و ۱۵۰ اجرا خواهد شد. هر یک از ماشین‌های مجازی بار کاری تصادفی بین ۱۰۰۰ تا ۱۰۰۰۰ دارند تا تنوع ماشین‌ها تضمین شود. نیمه اول هر ماشین مجازی سرعت پردازشی ۲۰۰۰ و نیمه دوم ۴۰۰۰ خواهد داشت. هزینه‌های نیمه اول به ازای هر واحد زمانی ۲۰۰ دلار و هزینه‌های نیمه دوم ۴۰۰ دلار تعیین شده است. برای بررسی تأثیر افزایش تعداد وظایف و ماشین‌های مجازی بر روی معیارهای عملکرد مختلف (زمان انجام، مصرف انرژی، هزینه‌ها، زمان جریان، نرخ انتشار دی‌اکسید کربن و کارایی)، از این دو مجموعه داده استفاده شده است.

۹-۱- فعالیت‌های پیش‌رو

در ادامه‌ی این مطالعه می‌توان کارایی MPA و الگوریتم‌های بهبود یافته آن را در محیط‌هایی دارای وابستگی‌های وظایف که به طور پویا در حال تغییر است بررسی کرد. پیاده‌سازی وابستگی‌های پویای وظیفه شامل ایجاد مدلی است که با در نظر گرفتن وابستگی‌های متقابل وظایف بتولند برنامه‌ریزی وظایف را در زمان واقعی تنظیم کند. این رویکرد به منظور بهینه‌سازی تخصیص منابع، کاهش تأخیر و بهبود عملکرد کلی سیستم در محیط‌های پویا و غیرقابل پیش‌بینی است. اجرای آن به دلیل نیاز به تصمیم‌گیری سریع و پیچیدگی روابط وابستگی، چالش‌برانگیز است. با انجام پژوهش و پیاده‌سازی این ایده می‌توان از سازگاری بلادرنگ و بهینه‌سازی منابع توسط این الگوریتم اطمینان حاصل کرد و نتایج را با الگوریتم‌های دیگر در این زمینه مقایسه کرد.

همچنین می‌توان MPA را با سایر تکنیک‌های بهینه‌سازی برای رسیدگی به سناریوهای زمان‌بندی پیچیده که شامل وظایف و منابع ناهمگن است ترکیب کرد. یا در مساله‌های امنیتی ورود کرد و نحوه ادغام MPA با پروتکل های امنیتی برای اطمینان از زمان بندی وظایف ایمن در مسائل اینترنت اشیاء را بررسی کرد. بررسی عملکرد این الگوریتم‌ها در کاربردهای خاص منظوره‌تر مانند مسائل پزشکی یا شهری نیز می‌تواند در ادامه‌ی مسیر ایده‌ی مناسبی باشد.

۲- پیاده‌سازی زمان‌بندی

محاسبات مه، گسترشی از محاسبات ابری است که در آن منابع محاسباتی در لبه شبکه و نزدیک‌تر به منابع داده یا کاربران نهایی قرار می‌گیرند. iFogSim یک ابزار طراحی شده برای شبیه‌سازی و مدل‌سازی محیط‌ها و برنامه‌های محاسبات مه است. iFogSim به عنوان توسعه‌ای بر CloudSim معرفی شده است تا قابلیت‌های محاسبات مه را به آن اضافه کند. در حالی که CloudSim برای شبیه‌سازی محیط‌های محاسبات ابری طراحی شده است، iFogSim

تمرکز خود را بر روی چالش‌ها و ویژگی‌های خاص محاسبات مه قرار داده است. این ابزار به محققان و مهندسان اجازه می‌دهد تا برنامه‌ریزی منابع، تخصیص وظایف، و سیاست‌های مدیریت انرژی را در محیط‌های محاسبات مه مدل‌سازی و تحلیل کنند.

iFogSim با ارائه یک معماری چندسطحی، امکان شبیه‌سازی و تعامل بین دستگاه‌های لبه[۱]، گره‌های مه[۲] و مراکز داده ابری را فراهم می‌کند. این ویژگی اجازه می‌دهد تا بر روی توزیع هوشمندانه منابع و کاهش تاخیر در پردازش داده‌ها در محیط‌های محاسبات مه تمرکز کنیم. iFogSim همچنین امکان مدل‌سازی و تحلیل جریان داده‌ها و پردازش‌های محلی در لبه شبکه[۳] را فراهم می‌کند، که این امر برای برنامه‌های کاربردی مانند اینترنت اشیاء [۴] بسیار کاربردی است.

ماشین‌های مجازی[۵] در iFogSim، منابع محاسباتی مجازی را نمایندگی می‌کنند که می‌توانند به خدمات و برنامه‌های مختلف در شبکه محاسبات مه اختصاص یابند. این ابزار از ماشین‌های مجازی برای شبیه‌سازی فرآیند تخصیص منابع در محیط مه استفاده می‌کند، شامل CPU، حافظه و ذخیره‌سازی و غیره.

با استفاده از ماشین‌های مجازی، iFogSim به محققان و توسعه‌دهندگان امکان می‌دهد تا استراتژی‌های مختلف تخصیص منابع را آزمایش کنند و عملکرد آن‌ها را ارزیابی کنند، که می‌تولند شامل جنبه‌هایی مانند زمان پاسخ، مصرف انرژی و کارایی باشد. ماشین‌های مجازی انعطاف‌پذیری را برای افزایش یا کاهش منابع محاسباتی بر اساس تقاضا فراهم می‌کنند، که در iFogSim برای شبیه‌سازی سناریوهای واقعی که در آن‌ها نیازهای کاری و منابع می‌توانند به طور قابل توجهی متفاوت باشند، حیاتی است. ماشین‌های مجازی در iFogSim به کاربران امکان می‌دهند تا سناریوها و موارد استفاده مختلفی برای محاسبات مه ایجاد کنند، از جمله برنامه‌های اینترنت اشیاء، پیاده‌سازی‌های شهر هوشمند، سیستم‌های نظارت بر سلامتی و غیره.

زمانی که مجموعه‌ای از وظایف و چندین ماشین مجازی با MIPS[۶] و مشخصات مختلف داریم، برنامه‌ریزی مؤثر وظایف حیاتی می‌شود. هدف این است که وظایف را به گونه‌ای به ماشین‌های مجازی اختصاص دهیم که اهداف خاصی مانند کاهش زمان اجرا، تعادل بار یا کاهش مصرف انرژی را بهینه‌سازی کند. یک رویکرد عمومی برای استفاده از ماشین‌های مجازی برای برنامه‌ریزی وظایف به این صورت است که، ابتدا نیازهای هر وظیفه را از نظر قدرت محاسباتی، حافظه و زمان اجرا مشخص می‌شود. سپس قابلیت‌های هر ماشین مجازی ارزیابی می‌شود، از جمله MIPS آن‌ها، حافظه موجود، پهنای باند شبکه و سایر ویژگی‌های مرتبط. سپس اهداف زمان‌بندی تعیین می‌شود. مانند کاهش حداکثر زمان اجرا، به حداکثر رساندن استفاده از منابع، کاهش مصرف انرژی یا تعادل بار بین ماشین‌های مجازی.

در نهایت یک الگوریتم زمان‌بندی بر اساس اهداف مطرح شده انتخاب می‌شود. در iFogSim، از الگوریتم‌های زمان‌بندی وظایف سفارشی پشتیبانی می‌شود و می‌توان الگوریتم انتخابی خود را در این چارچوب پیاده‌سازی کرد. بر اساس الگوریتم، وظایف به ماشین‌های مجازی نگاشت می‌شود. این مرحله تعیین می‌کند که با در نظر گرفتن عواملی مانند MIPS ماشین‌های مجازی، طول وظیفه و سایر نیازهای منابع کدام وظیفه باید روی کدام ماشین مجازی اجرا شود.

وظایف روی ماشین‌های مجازی اختصاص یافته اجرا می‌شوند و عملکرد آن‌ها قابل پایش است. این پایش شامل ردیابی زمان اجرا، استفاده از منابع و سایر معیارهای مرتبط است. در مراحل بعد بر اساس داده‌های عملکرد پیشین، می‌توان تغییراتی در استراتژی زمان‌بندی ایجاد کرد.

برای پیاده‌سازی الگوریتم‌های فراابتکاری ذکر شده در مقاله‌ی اولیه به منظور برنامه‌ریزی وظایف در iFogSim نیز باید چندین مرحله دنبال شود. از آن جایی که iFogSim به زبان جاوا است برای پیاده‌سازی رویکرد مد نظر نیز باید به زبان برنامه‌نویسی جاوا کدنویسی انجام شود. در اینجا یک چارچوب کلی از فرآیند ارائه شده است:

ابتدا باید محیط توسعه‌ی iFogSim فراهم شود. برای این کار باید JDK[۷] نصب شده باشد، همچنین یک محیط توسعه یکپارچه مانند Eclipse یا IntelliJ IDEA نیز برای پیاده‌سازی کد لازم است. سپس باید کد برنامه iFogSim را از مخزن Github مربوط به آن بارگیری کرده و درمحیط توسعه راه‌اندازی کرد.

وظایف و ماشین‌های مجازی داده‌های تستی تعریف می‌شوند. یک تابع برای تعریف وظایف[۸] با طول‌های مشخص ایجاد می‌شود. تابع دیگری نیز برای ایجاد ماشین‌های مجازی با MIPS و مولفه‌های مورد نیاز تعریف می‌شود. الگوریتم‌های مد نظر با زبان جاوا پیاده‌سازی می‌شود که توضیحات مربوط به آن در ادامه داده می‌شود.

الگوریتم با iFogSim یکپارچه می‌شود. کلاس‌های زمان‌بندی iFogSim برای استفاده از الگوریتم فراابتکاری برای تخصیص وظیفه تغییر یا گسترش داده می‌شود. تنظیمات شبیه‌سازی انجام می‌شوند. در واقع یک شبیه‌سازی در iFogSim تنظیم می‌شود که در آن تعداد مورد نیاز وظایف و ماشین‌های مجازی ایجاد شده است. وظایف نیز با استفاده از الگوریتم زمان‌بندی پیاده‌سازی شده به ماشین‌های مجازی اختصاص داده می‌شوند. شبیه‌سازی اجرا می‌شود و نتایج زمان‌بندی مانند زمان اجرا و هزینه و.. را گزارش می‌دهد. داده‌های جمع‌آوری شده

---

۱ Edge Devices
۲ Fog Nodes
۳ Edge Processing
۴ IoT
۵ VMs
۶ میلیون دستورالعمل در ثانیه
۷ Java Development Kit
۸ Cloudlets

تجزیه و تحلیل می‌شـــوند تا کارایی و اثربخشـــی الگوریتم برنامه‌ریزی محک بخورد.

در محیط CloudSim، دو مفهوم کلیـدی وجود دارد کـه در شبیه‌سازی زیرساخت‌های ابری نقش بسیار مهمی ایفا می‌کنند و از آن جایی که iFogSim توســعه‌ای بر CloudSim اســت همچنان مورد توجه ما است: مراکز داده [1] و کارگزاران[2].

مراکز داده در CloudSim نماینده‌ی زیرســاخت‌های فیزیکی در محیط ابری هستند. این مراکز شامل مجموعه‌ای از منابع سخت‌افزاری مانند واحدهای پردازشـی[3]، حافظه، پهنای باند و فضـای ذخیره‌سـازی هستند. هر مرکز داده می‌تواند چندین ماشین میزبان [4] داشته باشد که هر کدام شامل چندین واحد پردازشی و منابع دیگر است.

مراکز داده در CloudSim برای شـــبیه‌ســـازی محیط‌های ابری واقعی بسیار مهم هستند. آن‌ها می‌توانند پیکربندی‌های متفاوتی داشته باشند که نشان‌دهنده‌ی انواع مختلف مراکز داده در دنیای واقعی است. این پیکربندی‌ها شامل مشخصات سخت‌افزاری، سیاست‌های تخصیص منابع و استراتژی‌های مدیریت انرژی است.

کارگزاران در CloudSim نقش مدیریت و تخصـــیص منابع را بر عهده دارند. کارگزار یک نماینده برای کاربران نهایی است که تقاضاهای منابع را مدیریت می‌کند. این تقاضـاها معمولاً به صـورت ماشـین‌های مجازی و وظایف محاسباتی هستند.

کارگزاران وظیفه دارند تا درخواســت‌های منابع را از کاربران دریافت کرده و بر اسـاس سـیاسـت‌های تعریف شـده، این منابع را در ماشین‌های میزبان موجود در مراکز داده تخصیص دهند. آن‌ها همچنین مســئولیت نظارت بر اجرای وظایف و مدیریت عملکرد ماشـــین‌های مجازی را بر عهده دارند.

الگوریتم‌های زمانبندی مورد بررسی باید در قالب کارگزاران پیاده سازی شوند تا

فایل‌های مربوط به این فاز از انجام پروژه به منظور پیاده ســـازی الگوریتم زمـان‌بنـدی MPA و اعمـال دو مجموعـه داده از وظـایف و ماشین‌های مجازی به صورت زیرند:

Constants.java:

این فایل شامل تعاریف ثابت‌هایی است که در برنامه جاوا استفاده می‌شـــوند. در این فایل، ثابت‌ها به صـــورت متغیرهای public static final تعریف شـده‌اند که به این معناسـت که این متغیرها در سـراسـر برنامه قابل دسـترسـی هسـتند، تغییر ناپذیرند و مقدار آن‌ها در زمان کامپایل تعیین می‌شود.

این پارامترها شامل مواردی مانند تعداد ماشین‌های مجازی، طول وظایف، جمعیت اولیه، حداکثر تکرار و دیگر مقادیر ثابت که در طول اجرای الگوریتم تغییر نمی‌کنند اسـت. اسـتفاده از فایل جداگانه برای تعریف ثابت‌ها به مدیریت بهتر کد کمک می‌کند. این روش به تضـمین اینکه تمام بخش‌های برنامه از مقادیر ثابت یکسـانی اسـتفاده می‌کنند کمک می‌کند و خطاهای ناشـی از اسـتفاده نادرسـت یا تعریف مجدد مقادیر ثابت را کاهش می‌دهد.

Common.java:

توابعی که برای محاسبه‌ی پارامترهای مختلف و تابع تناسب اسـتفاده شـده بودند مطابق با فرمول هایی که در قسـمت‌های پیشـین آوردیم در این فایل پیاده‌سازی شده اند. همچنین برخی محاسبات پر کاربر نیز در قالب توابع در اینجا پیاده‌سازی شدند.

MPABroker.java:

این کلاس یک توســعه از DatacenterBroker در CloudSim اســـت و بخش اصـــلی الگوریتم MPA را پیـاده‌ســـازی می‌کنـد. MPABroker نقش یک کارگزار را در محیط CloudSim ایفا می‌کند که مسـئولیت تخصـیص منابع و زمان‌بندی وظایف را بر عهده دارد. در این فایل، مکانیزم‌های مربوط به الگوریتم MPA مانند جســتجوی فضـای حل، بهینه‌سـازی تخصـیص وظایف، و مدیریت وابسـتگی‌های وظیفه‌ای ممکن است پیاده‌سازی شده باشند. این فایل همچنین شامل توابعی برای ارزیابی عملکرد و مقایسه راه‌حل‌های پیدا شده است.

Algorithm: Submit Cloudlets
Input: List of cloudlets, List of VMs
Output: Submission of cloudlets based on the optimized solution

```
1: Initialize predators list
2: Set bestFitness to Double.MAX_VALUE
3: Set bestSolution to null

4: for iter = 0 to MAX_ITERATION do
5:     for each predator in predators do
6:         fitness = evaluateFitness(predator)
7:         predatorFitnessMap.put(predator, fitness)
8:         if fitness < bestFitness then
9:             bestFitness = fitness
10:            bestSolution = clone of predator
11:        end if
12:    end for

13:    UpdatePredators(predators, iter)
14: end for

15: if bestSolution is not null then
16:     SubmitCloudletsBasedOnSolution(bestSolution)
17: end if
```

این الگوریتم با ایجاد لیسـتی از شـکارچیان آغاز می‌شـود که هر کدام نمایانگر یک راه‌حل احتمالی در فرآیند زمان‌بندی وظایف هستند. بهترین تناســب ابتدا به مقدار بی‌نهایت تعیین می‌شــود. در هر تکرار،

[1] Datacenters
[2] Brokers
[3] PEs
[4] Host

تناسب هر شکارچی ارزیابی شده و در صورت کمتر بودن نسبت به بهترین تناسب موجود، آن به عنوان بهترین راه‌حل ثبت می‌شود. در نهایت، بهترین راه‌حل یافته شده برای ارسال وظایف برای اجرا استفاده می‌شود.

در ابتدا، یک لیست از predators یا همان راه‌حل‌های بالقوه توسط الگوریتم مقداردهی اولیه می‌شوند. این راه‌حل‌ها نماینده‌ی تخصیص ممکن ابرداده‌ها به ماشین‌های مجازی هستند. در این مرحله، متغیر bestFitness به مقدار بسیار بزرگی تنظیم می‌شود و bestSolution خالی در نظر گرفته می‌شود. این کار برای آماده‌سازی محیط جستجوی بهینه انجام می‌شود.

سپس، الگوریتم وارد یک حلقه‌ی تکرار می‌شود که در هر تکرار آن، برازش هر یک از predators ها با استفاده از تابع evaluateFitness محاسبه می‌شود. این تابع برازش یک راه‌حل را بر اساس معیارهای مشخص شامل زمان اتمام و مصرف ارزیابی می‌کند. در صورتی که برازش یک predator از بهترین برازش تاکنون کمتر باشد، این predator به عنوان بهترین راه‌حل جدید در نظر گرفته می‌شود.

پس از اتمام تمام تکرارها، در صورتی که bestSolution خالی نباشد، الگوریتم ابرداده‌ها را بر اساس بهترین راه‌حل یافت شده به ماشین‌های مجازی اختصاص می‌دهد. این کار با استفاده از تابع SubmitCloudletsBasedOnSolution انجام می‌شود که ابرداده‌ها را بر اساس تخصیص مشخص شده در bestSolution به ماشین‌های مجازی مرتبط می‌کند.

Procedure: initializePredators()
Input: Lists of cloudlets and VMs
Output: Initialized list of predators

```
1: Get numTasks as the size of cloudletList
2: Get numVMs as the size of vmList
3: for I = 0 to PREDATORS_NO do
4:     Initialize predator array of size numTasks
5:     for j = 0 to numTasks do
6:         predator[j] = random integer from 0 to numVMs - 1
7:     end for
8:     Add predator to predators list
9:     Print predator and its fitness
10: end for
```

در این رویه، ابتدا تعداد وظایف و ماشین‌های مجازی مشخص می‌شوند. سپس برای هر شکارچی، مقادیر اولیه به صورت تصادفی بر اساس تعداد ماشین‌های مجازی انتخاب می‌شوند. این فرآیند باعث ایجاد تنوع در راه‌حل‌های اولیه می‌گردد و به الگوریتم کمک می‌کند تا فضای جستجو را به طور کامل پوشش دهد.

Procedure: **SubmitCloudletsBasedOnSolution**(solution)
```
1: for i = 0 to size of cloudletList do
2:     vmId = solution[i]
3:     Bind cloudlet i to vmId
4: end for
5: Report results
6: Call super.submitCloudlets()
```

این رویه به ارسال وظایف بر اساس بهترین راه‌حل یافته شده می‌پردازد. هر وظیفه بر اساس این راه‌حل به یک ماشین مجازی مشخص ارتباط داده می‌شود. در نهایت، نتایج حاصل از این ارسال گزارش می‌گردد.

Procedure: **UpdatePredators**(predators, currentIteration)
```
1: for each predator in predators do
2:     if currentIteration < maxIteration / 3 then
3:         Apply BrownianMotion to each element of predator
4:     else if currentIteration between maxIteration / 3 and 2 *
maxIteration / 3 then
5:         Apply BrownianMotion to first half and LevyFlight to second
half of predator
6:     else
7:         Apply LevyFlight to each element of predator
8:     end if
9: end for
```

در این رویه، بر اساس مرحله‌ای که الگوریتم در آن قرار دارد (کاوش، تعادل بین کاوش و بهره‌برداری، یا بهره‌برداری)، شکارچیان به‌روزرسانی می‌شوند. در مرحله کاوش، از حرکت براونی استفاده می‌شود؛ در مرحله تعادل، نیمی از شکارچیان به وسیله حرکت براونی و نیم دیگر به وسیله پرواز لوی به‌روزرسانی می‌شوند؛ و در مرحله بهره‌برداری، همه شکارچیان از طریق پرواز لوی به‌روز می‌شوند.

Procedure: LevyFlight(predator, index)
Input: predator - an array representing a solution, index - the position to update
Output: Updated position of the predator at the given index

```
1: Set beta to 1.5
2: Calculate sigma using gamma function and beta
3: Generate random values u and v using Gaussian distribution
4: Calculate step using u, v, and beta
5: newPosition = predator[index] + step
6: Return clamp(newPosition)
```

روش LevyFlight برای اعمال یک حرکت تصادفی با قدم‌هایی با طول توزیع نامنظم استفاده می‌شود. در این روش، ابتدا مقداری برای پارامتر beta تعیین می‌شود که در اینجا ۱.۵ است. سپس، با استفاده از تابع گاما و مقدارbeta ، یک مقدار برای sigma محاسبه می‌شود. پس از آن، دو مقدار تصادفی u و v با استفاده از توزیع گاوسی تولید می‌شوند. مقدار قدم نهایی با استفاده از این دو مقدار و beta محاسبه می‌شود. در نهایت، این قدم به موقعیت فعلی predator در اندیس مورد نظر اضافه می‌شود تا موقعیت جدید به دست آید.

Procedure: BrownianMotion(predator, index)
Input: predator - an array representing a solution, index - the position to update
Output: Updated position of the predator at the given index

```
1: Set stepSize to a small value (e.g., 0.1)
2: Generate a random Gaussian value gaussian
3: newPosition = predator[index] + (stepSize * gaussian)
4: Return clamp(newPosition)
```

در روشBrownianMotion ، یک حرکت تصادفی ساده‌تر با قدم‌های کوچکتر انجام می‌شود. در این روش، ابتدا اندازه قدم به یک

مقدار کوچک (مانند ۰.۱) تنظیم می‌شود. سپس، یک مقدار تصادفی گاوسی تولید می‌شود. این مقدار تصادفی با اندازه قدم ضرب شده و به موقعیت فعلی predator در اندیس مورد نظر اضافه می‌شود تا موقعیت جدید حاصل شود.

MMPABroker.java:

Algorithm: Submit Cloudlets using MMPA

Input: List of cloudlets, List of VMs

Output: Submission of cloudlets based on the optimized solution

1: Initialize predators list

2: Set bestFitness to Double.MAX_VALUE

3: Set bestSolution to null

4: for iter = 0 to MAX_ITERATION do

5: for each predator in predators do

6: fitness = evaluateFitness(predator)

7: predatorFitnessMap.put(predator, fitness)

8: if fitness < bestFitness then

9: bestFitness = fitness

10: bestSolution = clone of predator

11: end if

12: end for

13: if iter % PredefinedInterval == 0 then

14: ReInitializeHalfPopulation(predators)

15: end if

16: UpdatePredators(predators, iter)

17: end for

18: if bestSolution is not null then

19: SubmitCloudletsBasedOnSolution(bestSolution)

20: end if

IMMPABroker.java:

1: Initialize predators list with random solutions

2: Set bestFitness to a high value

3: Set bestSolution to null

4: for iter = 0 to MAX_ITERATION do

5: Evaluate fitness of each predator and update best solution

6: Apply ranking-based reinitialization and mutation

7: if iter % PIT == 0 then

8: Reinitialize half of the predators randomly

9: end if

10: Update predators based on IMMPA algorithm

11: Mutate predators towards the best solution

12: end for

13: if bestSolution is not null then

14: SubmitCloudletsBasedOnSolution(bestSolution)

15: end if

TaskSchedulingSimulation.java

این فایل به عنوان هسته اصلی شبیه‌سازی عمل می‌کند و تمام اجزای مختلف پروژه را به یکدیگر متصل می‌کند تا یک شبیه‌سازی دقیق و کارآمد از محیط ابری و برنامه‌ریزی وظایف ایجاد شود. TaskSchedulingSimulation مسئولیت راه‌اندازی محیط شبیه‌سازی، ایجاد وظایف و منابع مورد نیاز، و اجرای الگوریتم‌های زمان‌بندی را بر عهده دارد. این فایل همچنین شامل بخش‌هایی برای جمع‌آوری و تجزیه و تحلیل داده‌های عملکرد شبیه‌سازی است، که این امکان را می‌دهد تا کارایی و اثربخشی الگوریتم‌ها در شرایط مختلف ارزیابی شود.

Algorithm: TaskSchedulingSimulation
Input: None
Output: Simulation Results

Begin
  Initialize CloudSim
    numUser ← 1
    calendar ← getCurrentInstance()
    traceFlag ← false
    CloudSim.init(numUser, calendar, traceFlag)

  Create first Datacenter
    datacenter0 ← createDatacenter()

  Create first MPA Broker
    broker ← createMPABroker("MPA_Broker")

```
    brokerId ← if broker ≠ null then broker.getId() else 0

  Create first set of VMs
    vmList ← createVM(brokerId)
    if broker ≠ null then
      broker.submitVmList(vmList)

  Create first set of Cloudlets
    cloudletList ← createCloudlet(brokerId)
    if broker ≠ null then
      broker.submitCloudletList(cloudletList)

  Start first simulation
    CloudSim.startSimulation()

  Stop first simulation
    CloudSim.stopSimulation()

  Retrieve and print results of first simulation
    resultList ← broker.getCloudletReceivedList()
    printCloudletList(resultList, vmList)

  Reinitialize CloudSim for second simulation
    CloudSim.init(numUser, calendar, traceFlag)
    datacenter1 ← createDatacenter()

  Create second MPA Broker
    broker2 ← createMPABroker("MPA_Broker2")
    brokerId2 ← if broker2 ≠ null then broker2.getId() else 0

  Create second set of VMs
    vmList2 ← createVM2(brokerId2)
    if broker2 ≠ null then
      broker2.submitVmList(vmList2)

  Create second set of Cloudlets
    cloudletList2 ← createCloudlet2(brokerId2)
    if broker2 ≠ null then
      broker2.submitCloudletList(cloudletList2)

  Start second simulation
    CloudSim.startSimulation()

  Stop second simulation
    CloudSim.stopSimulation()

  Retrieve and print results of second simulation
    resultList2 ← broker2.getCloudletReceivedList()
    printCloudletList(resultList2, vmList2)

End
```

فرآیند اجرای کد و وظایف هر تابع در TaskSchedulingSimulation به شرح زیر است:

در ابتدای تابع main، CloudSim با تعداد کاربران، تقویم و پرچم ردیابی[1] مقداردهی می‌شود. این گام زمینه‌ای برای اجرای شبیه‌سازی فراهم می‌کند. سپس، با استفاده از توابع createDatacenter و createMPABroker، مراکز داده و کارگزاران ایجاد می‌شوند. کارگزار MPA برای مدیریت و توزیع وظایف و منابع مجازی به کار می‌رود.

با استفاده از توابع createVM و createCloudlet، لیست‌هایی از ماشین‌های مجازی و وظایف ایجاد می‌شوند. این‌ها نشان‌دهنده وظایفی هستند که باید در مراکز داده اجرا شوند.

با فراخوانی CloudSim.startSimulation، شبیه‌سازی آغاز می‌شود. این گام شامل اجرای وظایف مختلف در ماشین‌های مجازی و مدیریت منابع توسط کارگزاران است. پس از پایان شبیه‌سازی با CloudSim.stopSimulation، نتایج به دست آمده از وظایف اجرا شده توسط تابع printCloudletList چاپ می‌شوند. این نتایج شامل اطلاعاتی مانند شناسه وظایف، وضعیت، شناسه مرکز داده و ماشین مجازی اختصاص داده شده، زمان شروع و پایان اجرا و غیره هستند.

تابع exportCloudletList برای ارسال جزئیات وظایف به یک فایل اکسل استفاده می‌شود. اطلاعات مربوط به هر وظیفه در سلول‌های مختلف ذخیره می‌شوند.

همچنین توابع createCloudlet2 و createVM2 شبیه توابع createCloudlet و createVM هستند اما با پارامترها و تنظیمات متفاوت برای ایجاد سری دوم وظایف و ماشین‌های مجازی.

۳- کارهای قبلی انجام شده

در این بخش، ضمن بررسی پژوهش‌های پیشین مرتبط با برنامه‌ریزی وظایف در محاسبات مه، به تحلیل مزایا و معایب الگوریتم‌های اصلی مطرح شده در هر پژوهش خواهیم پرداخت و اطلاعاتی ارائه خواهیم داد.

وو و همکاران [5] با توسعه یک الگوریتم برنامه‌ریزی برای کاهش مصرف انرژی، رویکردی نوآورانه ارائه دادند. مزیت این رویکرد در استفاده از مدل برنامه‌ریزی خطی صحیح برای شناسایی عوامل کلیدی کاهش مصرف انرژی است. با این حال، معایب آن شامل پیچیدگی بالای محاسباتی و محدودیت در مقیاس‌پذیری در محیط‌های بزرگ‌تر است.

لی و همکاران[6] با پیشنهاد چارچوب مه مجازی برای غلبه بر محدودیت‌های منابع در سطح گره‌های حسگری، راهکاری جامع ارائه دادند. این رویکرد به‌ویژه در ایجاد برنامه‌های سفارشی برای کاربران نهایی مفید است. با این حال، این چارچوب ممکن است در مواجهه با تنوع بالای تقاضاها و نیازهای مختلف کاربران، با چالش‌هایی روبرو شود.

کار خطاک و همکاران [7] که در آن از منابع به طور موثر با تعادل بار بین سرورها برای کاهش تاخیر و اضافه بار ترافیکی استفاده شده است، به‌خوبی نشان‌دهنده اهمیت مدیریت منابع در محاسبات مه است. این رویکرد به‌ویژه برای کاهش تاخیر در برنامه‌های حساس به زمان مانند خدمات بهداشتی مفید است. با این حال، چالش اصلی در این رویکرد، توزیع عادلانه بار کاری بین سرورهای مختلف و جلوگیری از اضافه بار بر روی سرورهای خاص است.

لی و همکاران [8] با ارائه روش‌هایی بر اساس فرآیند تصمیم‌گیری نیمه‌مارکوف برای هماهنگی روش‌های تخصیص ماشین مجازی، یک دیدگاه متفاوت در مدیریت منابع ارائه دادند. این رویکرد با تمرکز بر هماهنگی منابع، به‌ویژه در موقعیت‌هایی که محدودیت‌های منابع وجود

[1] trace flag

دارد، موثر اســـت. با این حال، این روش‌ها ممکن اســـت در مواجهه با تغییرات ناگهانی در تقاضا و پویایی محیط، با چالش‌هایی روبرو شوند.

جیانگ و همکاران [9] با ارائه یک مکانیســـم موثر از نظر انرژی و فرســـتنده انتقال بار، روی کنترل موثر منابع محاســـباتی تمرکز کردند. این رویکرد بـه‌ویژه برای کنترل بهینـه منـابع در محیط‌هـای بـا منـابع محدود مفید اســـت. با این حال، اجرای این مکانیســـم در محیط‌های بزرگ‌تر و پیچیده‌تر ممکن است به چالش‌هایی منجر شود.

دنگ و همکاران [10] با پیشـــنهاد مدل خود برای توزیع بار کاری در محاســـبات مه-ابری، تلاش کردند تا تاخیر انتقال و مصـــرف برق را کاهش دهند. با اینکه این مدل در کاهش هزینـه‌های عملیاتی موثر است، اما ممکن است در مواجهه با حجم بالای ترافیک و تقاضای متغیر کاربران IoT کمی محدود باشد. الگوریتم‌های مانند MPA می‌توانند با قابلیت تطبیق‌پذیری بالای خود، در این زمینه‌ها راه‌حل‌های بهینه‌تری ارائه دهند.

در [11]، مســـئله تخصـــیص خدمات به عنوان یک کوله‌پشـــتی چندبعدی فرموله شـــده و به دنبال راه‌حل‌های بهینه برای تخصـــیص خدمات با در نظر گرفتن تاخیر، تعادل بار و مصـــرف انرژی اســـت. این رویکرد بـه‌خوبی نشـــان‌دهنـده اهمیت توازن بین عوامل مختلف در تخصیص منابع است. با این حال، پیاده‌سازی و محاسبات مربوط به این مدل ممکن اســـت در محیط‌های بزرگ و پیچیده با چالش‌هایی روبرو شوند.

در [12]، برنامه‌ریزی تسک در سیستم محاسبات مه با استفاده از یـک الگوریتم مبتنی بر ابتکـاری برای تعـادل بین زمـان اتمـام کلی و هزینه‌های مالی محاســـبات ابری مورد بررســـی قرار گرفته اســـت. این الگوریتم به‌خوبی تعادل بین هزینه و کارایی را در محیط‌های محاسبات مه نشان می‌دهد. با این حال، این رویکرد ممکن است در موقعیت‌هایی که نیاز به تصمیم‌گیری‌های سریع و دقیق وجود دارد، با محدودیت‌هایی روبرو شود.

ین و همکاران [13] با ارائه یک مدل برنامه‌ریزی تســـک که نقش کانتینرها را در نظر می‌گیرد و همچنین یک الگوریتم برنامه‌ریزی تسک را برای اطمینان از اجرای به موقع تســـک‌ها و بهبود تعداد تســـک‌های همزمان برای محاســـبات مه می‌ســـازد، رویکردی جامع را ارائه داده‌اند. این مدل به‌خوبی نشـان‌دهنده اهمیت بهینه‌سـازی زمانبندی و مدیریت منابع در محاســـبات مه اســـت. با این حال، پیچیدگی مدل و نیاز به تطابق دقیق با نیازهای واقعی کاربران می‌تولند به عنوان چالش‌های این رویکرد در نظر گرفته شود.

تران و همکاران [14] با ارائه یک رویکرد برای بهبود عملکرد IoT از نظر زمان پاسخ، هزینه و انرژی، راه‌حل‌های نوآورانه‌ای را ارائه داده‌اند. این رویکرد تأکید ویژه‌ای بر بهینه‌سازی عملکرد و کاهش هزینه‌ها دارد. با این حال، چالش اصلی در این رویکرد، تطابق با تنوع و پویایی تقاضاها در محیط‌های IoT است.

ژو و همکاران [15] با ارائه مدلی برای تعادل بار در محاسبات مه-ابری، رویکردی مهم در جهت بهبود تعادل بار کاری ارائه داده‌اند. این مدل با تمرکز بر تعادل بار و کاهش تاخیر، به ارائه راه‌حل‌های موثر برای مدیریت منابع می‌پردازد. با این حال، این مدل ممکن اســـت در مواجهه با تغییرات سریع و پیچیده در تقاضاهای کاربری، با چالش‌هایی روبرو شود.

رهبری و همکاران [16] با پیشـــنهاد یک الگوریتم برنامه‌ریزی مبتنی بر کوله‌پشتی طماعانه برای اختصاص مناسب منابع به ماژول‌ها در محاسبات مه، رویکردی موثر در جهت بهینه‌سازی تخصیص منابع ارائه داده‌اند. این الگوریتم به‌خوبی توانایی تخصیص منابع را با تمرکز بر بهره‌وری و کارایی نشان می‌دهد. با این حال، این الگوریتم ممکن است در موقعیت‌هایی با تقاضـاهای پیچیده و متغیر، با محدودیت‌هایی روبرو شود.

الگوریتم‌های متاهیوریســـتیک نقش حیاتی در حل بســـیاری از مسائل واقعی دارند و در حل مسائل برنامه‌ریزی وظایف در محاسبات مه نیز به کار گرفته شده‌اند. بیتام و همکاران [17] یک الگوریتم زندگی زنبورها را برای مقابله با مشکل برنامه‌ریزی کار در محاسبات مه توسعه دادند که به دنبال یافتن بهترین تعادل بین زمان اجرای CPU و حافظه اختصـاص یافته مورد نیاز توسـط محاسـبات مه اسـت. این روش در کاهش زمان اجرا و بهینه‌سازی استفاده از حافظه موثر است، اما ممکن است در محیط‌های با پیچیدگی بالا با چالش‌هایی مواجه شود. گررو و همکاران [18] عملکرد سه الگوریتم تکاملی را برای بهبود تاخیر شبکه، بهره‌برداری از منابع و خدمات اختصـاص داده شـده مورد بررسـی قرار دادند. این رویکردها در بهینه‌سازی تاخیر و کارایی منابع موثر هستند، اما ممکن اســـت در برابر تغییرات پویای محیط‌های محاســـباتی انعطاف‌پذیری کمتری داشته باشند.

رهبری و نیکرای[19] از الگوریتم جســـتجوی ســـمبیوتیک برای برنامه‌ریزی تسک‌ها در محاسبات مه بر اساس یک کوله پشتی استفاده کردند. این روش در تخصـیص بهینه منابع و جلوگیری از اضـافه بار بر روی یک گره خاص موثر است. در [20] ، الگوریتم‌های جستجوی کوکو و گرده‌افشانی گل‌ها برای متعادل کردن بار کاری درخواست‌های ارسالی به ابر و حل مشـــکل تاخیر و تاخیر ناشـــی از ابر و بهبود عملکرد مه پیشـنهاد شـده اسـت. این الگوریتم‌ها در کاهش تاخیر و بهبود توزیع بار کاری موثر هستند، اما ممکن است در شرایطی که نیاز به تصمیم‌گیری سریع وجود دارد، کمتر کارآمد باشند.

در [21]، بهینه‌ســـازی کلونی مورچه‌ها (ACO) و بهینه‌ســـازی ازدحام ذرات (PSO) برای ارائه دو الگوریتم برنامه‌ریزی پیشـنهاد شـده اســـت که بار کاری را بر روی گره‌های مه با در نظر گرفتن هزینه‌های ارتباطی و زمان پاســـخ تعادل بخشـــیده‌اند. این روش‌ها در کاهش هزینه‌های ارتباطی و توزیع متعادل بار کاری موثرند، اما ممکن است در مواجهـه بـا تغییرات دینـامیکی در درخواســـت‌هـا و بـار کـاری دچـار محـدودیـت‌هـایی شـــونـد. در [22]الگوریتم شـــعلـه موهای بـه عنوان

الگوریتمی برای برنامه‌ریزی وظایف پیشنهاد شده است. این الگوریتم در یافتن راه‌حل‌های بهینه برای برنامه‌ریزی تسک‌ها در محیط‌های پیچیده موثر است، اما ممکن است در شناسایی بهترین توزیع بار کاری در شرایط متغیر کمتر کارآمد باشد.

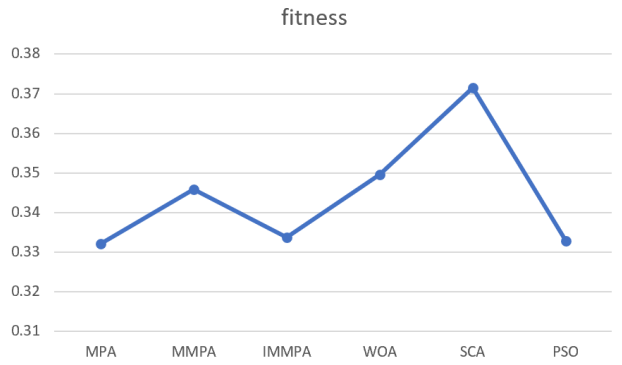
با این حال، علی‌رغم پیشنهاد بسیاری از الگوریتم‌ها، هنوز هم این الگوریتم‌ها موفق به تعادل بار کاری بین تمام ماشین‌های مجازی نشده‌اند. بنابراین، رویکرد جدیدی بر اساس رفتارهای الگوریتم شکارچیان دریایی هنگام حمله به طعمه‌های خود با برخی بهبودها به عنوان تلاشی برای رسیدن به کیفیت بهتر برای معیارهای عملکردی مانند مصرف انرژی، زمان اتمام کلی، زمان جریان کلی و نرخ انتشار دی اکسید کربن پیشنهاد شده است.

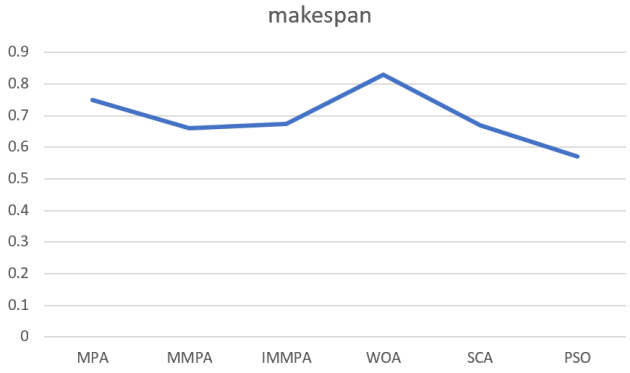
۴- نتایج

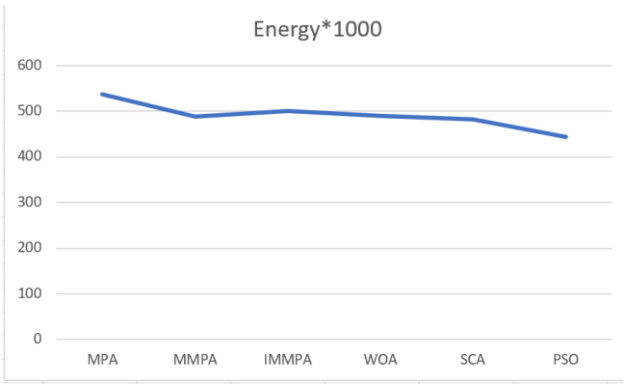
در این مرحله با پیاده‌سازی الگوریتم‌های ذکر شده در مقاله و اعمال مجموعه داده‌های مطرح شده به نتایجی رسیدیم که در زیر گزارش می‌شود. این نتایج حاصل ۵ دور تکرار آزمایش و میانگین گیری از نتایج آن است

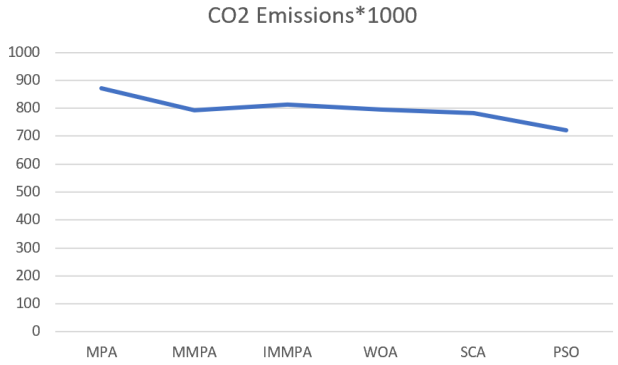
همان طور که مشاهده می‌شود الگوریتم IMMPA با تقریب مناسبی عملکرد مطلوبی دارد ولی الگوریتم‌های MPA و PSO نیز عملکرد نزدیکی به این مورد دارند.

مجموعه داده‌ی آزمایشی اول:

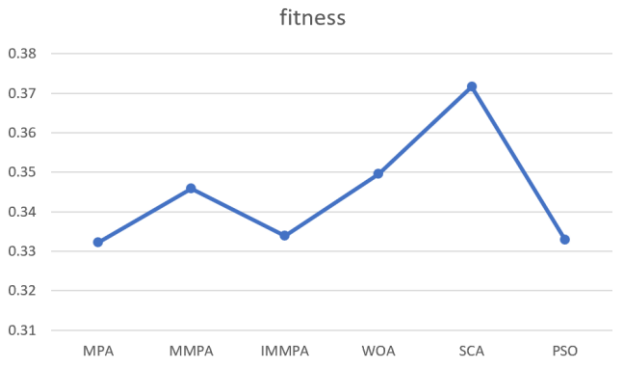


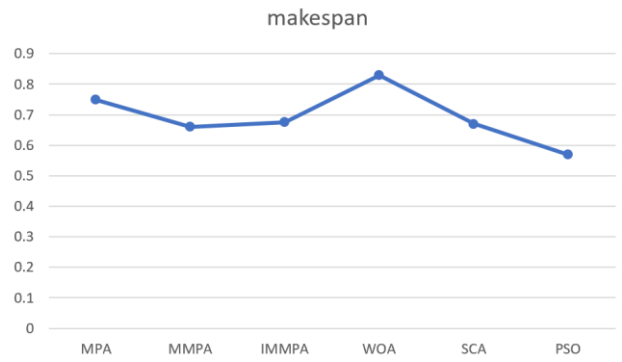


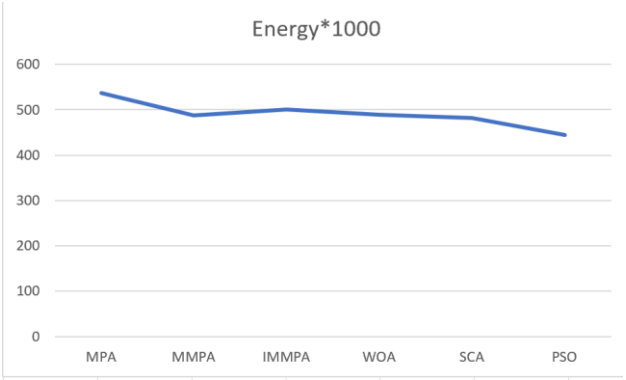


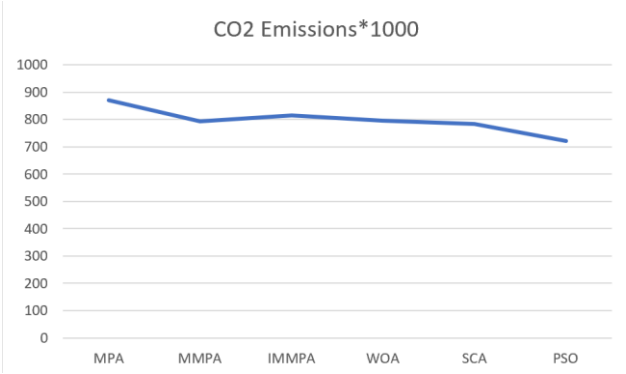


مجموعه داده‌ی آزمایشی دوم:









کدهای پیاده سازی و فایل مربوط به نتایج در پوشه‌ای مجزا الصاق شده است. به دلیل سنگین بودن فایل پروژه تنها فایل های مربوط به پیاده سازی آورده شده اند که برای اجرا باید در شبیه ساز در مسیر زیر جایگذاری شوند:

iFogSim\src\org\fog\test\perfeval\

فهرست مراجع

[1] R. D. K. D. A. e. a. Rai, "An Inclusive Survey on Marine Predators Algorithm: Variants and Applications," *Arch Computat Methods Eng*, p. 133–3172, 2023.

[2] R. M. S. M. R. K. C. a. M. R. M. Abdel-Basset, "An Efficient Marine Predators Algorithm for Solving Multi-Objective Optimization Problems: Analysis and Validations," *IEEE Access*, vol. 9, pp. 42817-42844, 2021.

[3] S. Mirjalili, "SCA: A Sine Cosine Algorithm for solving optimization problems," *Knowledge-Based Systems*, vol. 96, pp. 120-133, 2016.

[4] A. L. Seyedali Mirjalili, "The Whale Optimization Algorithm," *Advances in Engineering Software*, vol. 95, pp. 51-67, 2016.

[5] H.-Y. a. C.-R. L. Wu, "Energy efficient scheduling for heterogeneous fog computing architectures," *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 2018.

[6] J. e. a. Li, "Virtual fog: A virtualization enabled fog computing framework for Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 21-131, 2017.

[7] H. e. a. Khattak, "Utilization and load balancing in fog servers for health applications," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 91, 2019.

[8] Q. e. a. Li, "SMDP-based coordinated virtual machine allocations in cloud-fog computing systems," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1977-1988, 2018.

[9] Y.-L. e. a. Jiang, "Energy-efficient task offloading for timesensitive applications in fog computing," *IEEE Systems Journal*, vol. 13, no. 3, pp. 2930-2941, 2018.

[10] R. e. a. Deng, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE internet of things journal*, vol. 3, no. 6, pp. 1171-1181, 2016.

[11] V. e. a. Souza, "Towards distributed service allocation in fog-to-cloud (F2C)

scenarios," *IEEE global communications conference (GLOBECOM)*, 2016.

[12] X.-Q. a. E.-N. H. Pham, "Towards task scheduling in a cloud-fog computing system," in *18th Asia-Pacific network operations and management symposium(APNOMS)*, 2016.

[13] L. J. L. a. H. L. Yin, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4712-4721, 2018.

[14] M.-Q. e. a. Tran, "Task placement on fog computing made efficient for iot application provision," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[15] X. e. a. Xu, "A heuristic virtual machine scheduling method for load balancing in fog-cloud computing," in *IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing,(HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, 2018.

[16] D. a. M. N. Rahbari, "Low-latency and energyefficient scheduling in fog-based IoT applications," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 27, no. 2, pp. 1406-1427, 2019.

[17] S. S. Z. a. A. M. Bitam, "Fog computing job scheduling optimization based on bees swarm," *Enterprise Information Systems*, vol. 12, no. 4, pp. 373-397, 2018.

[18] C. I. L. a. C. J. Guerrero, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Generation Computer Systems*, vol. 97, pp. 131-144, 2019.

[19] D. a. M. N. Rahbari, "Scheduling of fog networks with optimized knapsack by symbiotic organisms search," in *21st Conference of Open Innovations Association (FRUCT)*, 2017.

[20] N. e. a. Javaid, "Cloud and Fog based Integrated Environment for Load Balancing using Cuckoo Levy Distribution and Flower Pollination for Smart Homes," in *2019 International Conference on Computer and Information Sciences (ICCIS)*, 2019.

[21] M. a. M. M. Hussein, "Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization," *IEEE Access*, vol. 8, p. 3719137201, 2020.

[22] M. e. a. Ghobaei-Arani, "An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 2, p. e3770, 2021.

[23] J. K. a. R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, WA, Australia, 1995.