دانشکده مهندسی کامپیوتر

الگوریتم تخلیه پردازش چندکاربره برای کمینه کردن تاخیر در دستگاه‌های اینترنت اشیا

پروژه‌ی پایانی کارشناسی مهندسی کامپیوتر

محمدمبین داریوش همدانی

استاد راهنما

رضا انتظاری ملکی

خرداد ۱۴۰۱


بسم الله الرحمن الرحيم

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پروژه

نام دانشکده: دانشکده مهندسی کامپیوتر

نام دانشجو: محمدمبین داریوش همدانی

عنوان پروژه : الگوریتم تخلیه پردازش چند کاربره برای کمینه کردن تاخیر در دستگاه‌های اینترنت اشیا

تاریخ دفاع: خرداد ۱۴۰۱

رشته: مهندسی کامپیوتر

ردیف	سمت	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه یا مؤسسه	امضـــــــــــا
۱	استاد راهنما	دکتر رضا انتظاری ملکی	استادیار	دانشگاه علم و صنعت ایران	
۲	استاد داور داخلی	دکتر	.....	دانشگاه علم و صنعت ایران	

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب محمدمبین داریوش همدانی به شماره دانشجویی ۹۶۵۲۱۱۹۱ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی تأیید می‌نمایم که کلیه‌ی نتایج این پروژه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری‌شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: محمدمبین داریوش همدانی

تاریخ و امضا:

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

□ بهره‌برداری از این پایان‌نامه برای همگان بلامانع است.

□ بهره‌برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.

□ بهره‌برداری از این پایان‌نامه تا تاریخ ..... ممنوع است.

استاد راهنما: رضا انتظاری ملکی

تاریخ:

امضا:

چکیده

رایانش لبه‌ای الگویی از محاسبات توزیع‌شده است که با نزدیک کردن منابع پردازشی به لبه شبکه، سعی دارد تا مزایایی مانند زمان پاسخگویی کمتر، مصرف باتری پایین‌تر و تحرک‌پذیری را برای کاربران به ارمغان بیاورد. از زمان معرفی رایانش لبه‌ای، یکی از چالش‌های مهم این حوزه، طراحی استراتژی‌های کارآمد برای تخلیه‌ی وظایف بوده است. علاوه بر این، با رشد روزافزون صنعت اینترنت اشیاء، تعداد زیادی کاربرد نرم‌افزاری جدید در سطح شبکه به وجود آمده است که هر کدام دارای نیازمندی‌های محاسباتی و شبکه‌ای خاص خود می‌باشند. بنابراین یک ویژگی مهم در طراحی استراتژی‌های تخلیه‌ی وظیفه در رایانش لبه‌ای، در نظر گرفتن ناهمگونی کاربردها از نظر میزان منابع مورد نیاز است. در پروژه‌ی فعلی روشی برای بدست آوردن استراتژی تخلیه‌ی وظیفه‌ی تاخیر-کمینه تحت محدودیت توان مصرفی ارائه می‌دهیم. روش پیشنهادی شامل دو قسمت می‌باشد. در قسمت اول، سیستم تخلیه‌ی وظایف را با کمک زنجیره مارکوف گسسته-زمان مدل‌سازی می‌کنیم و در قسمت دوم، با استفاده از الگوریتمی مبتنی بر برنامه‌ریزی خطی، استراتژی تخلیه بهینه را برای مدل ساخته شده محاسبه می‌کنیم. علاوه بر تشریح و حل مسئله به صورت تئوری، ساختار نرم‌افزاری جدیدی در زبان Kotlin ارائه می‌شود که می‌توان با استفاده از آن، استراتژی تخلیه‌ی بهینه را برای سیستم مورد نظر بدست آورد و عملکرد آن استراتژی را با کمک شبیه‌سازی بررسی کرد.

واژگان کلیدی: تخلیه‌ی وظیفه، رایانش لبه‌ای، زنجیره‌ی مارکوف، برنامه‌ریزی خطی، رایانش ابری

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست علایم اختصاری

حالت دستگاه کاربر . . . . . τ

تعداد وظایف موجود در صف i-اُم . . . . . q_i

نرخ ورود وظیفه به صف i-اُم . . . . . α_i

احتمال ارسال موفق بسته . . . . . β

مجموعه تمام حالت‌های دستگاه کاربر . . . . . S

مجموعه تمام کنش‌های ممکن . . . . . A

نسبت وظایف نوع i که محلی اجرا می‌شوند . . . . . η_i

توان مصرفی برای ارسال یک بسته . . . . . P_{tx}

توان مصرفی برای اجرای محلی به اندازه یک بازه زمانی . . . . . P_{loc}

حداکثر توان مصرفی قابل قبول . . . . . P_{max}

تعداد بازه زمانی لازم برای پردازش محلی وظایف نوع i . . . . . L_i

تعداد بازه زمانی لازم برای تخلیه‌ی وظایف نوع i . . . . . M_i

تعداد بازه زمانی لازم برای رایانش لبه‌ای وظایف نوع i . . . . . C_i

زمان اضافه لازم برای بازدریافت وظیفه از سرور لبه‌ای . . . . . t_{rx}

ظرفیت هر صف وظیفه . . . . . Q

تعداد قسمت اجرا شده از وظیفه تخصیص داده شده به پردازنده محلی . . . . . C_L

تعداد قسمت ارسال شده از وظیفه تخصیص داده شده به واحد ارسال . . . . . C_R

نوع وظیفه تخصیص داده شده به پردازنده محلی . . . . . T_L

نوع وظیفه تخصیص داده شده به واحد ارسال . . . . . T_R

فصل ۱

مقدمه

افزایش روز افزون تعداد دستگاه‌های موجود در لبه شبکه در سال‌های اخیر، و همچنین معرفی کاربردهای نرم افزاری جدید که نیازمند منابع محاسباتی بالا هستند باعث شده است که تقاضای زیادی برای خدمات پردازش ابری بوجود بیاید. پردازش ابری این امکان را به دستگاه‌های هوشمند از جمله تلفن همراه و اینترنت اشیاء می‌دهد که بخشی از پردازش‌های سنگین خود را به سرورهای قدرتمند «تخلیه» کنند تا بر محدودیت‌های پردازشی خود غلبه کنند و کاربردهای نرم افزاری پیچیده‌ای مانند واقعیت افزوده و خودروهای هوشمند را برای کاربران فراهم کنند.

با این وجود، پیاده‌سازی‌های سنتی پردازش ابری یک ایراد ذاتی دارند، و آن فاصله زیاد سرورهای ابری با دستگاه‌های پایانی است. الگوی «رایانش لبه‌ای» و معماری‌های استاندارد آن مانند رایانش لبه‌ای دسترسی-چندگانه که توسط سازمان ETSI ارائه شده است، سعی دارند تا با آوردن بخشی از منابع محاسباتی به نزدیکی لبه شبکه، این مشکل را تا حدی برطرف کنند. علاوه بر تمایل دستگاه‌های لبه شبکه به کمتر شدن این فاصله و به عبارتی «کشش» منابع محاسباتی توسط آن‌ها به منظور افزایش کیفیت سرویس، شرکت‌های ارائه‌دهنده خدمات ابری نیز تمایل دارند تا با «فشردن» بخشی از منابع محاسباتی خود به لبه شبکه، بار محاسباتی و هزینه‌های تجهیزاتی خود را کاهش دهند. [۲] در جدول ۱-۱ مقایسه‌ای کلی از رایانش ابری و لبه‌ای ارائه شده است. با دقت در این جدول می‌توان فهمید که رایانش لبه‌ای جایگزین رایانش ابری نمی‌باشد بلکه مکمل آن است.

ویژگی	رایانش ابری	رایانش لبه‌ای
استقرار	مرکزی	توزیع شده
فاصله تا دستگاه کاربر	زیاد	کم
تاخیر	زیاد	کم
تغییرات تاخیر[1]	زیاد	کم
منابع پردازشی	فراوان	محدود
فضای ذخیره‌سازی	فراوان	محدود

جدول ۱-۱: مقایسه رایانش ابری و لبه‌ای

یک امر مهم در پیاده‌سازی کارآمد رایانش لبه‌ای، طراحی استراتژی‌های تخلیه‌ی وظیفه به صورت هوشمند و موثر است. این استراتژی‌ها نحوه تخصیص منابع توسط دستگاه کاربر[2] را مشخص می‌کنند و این امکان را به دستگاه کاربر می‌دهند تا درباره تخلیه یا عدم تخلیه‌ی وظایف محاسباتی در طول زمان تصمیم بگیرد.

استراتژی تخلیه بهینه به استراتژی تخلیه‌ای گفته می‌شود که یک تابع «هدف» خاص را تحت شرایط محیطی مشخص کمینه یا بیشینه کند. توابع هدف عموما بر حسب یک یا چند معیار سیستم تعریف می‌شوند. برخی از این معیارها عبارتند از:

- تاخیر سرویس (Service Delay)
- توان مصرفی ((Consumed Power))
- تقدم و تاخر (Jitter)
- هزینه (Cost)

مقدار توابع هدف و شروط مسئله تخلیه بستگی به پارامترهای زیادی دارند، از جمله میزان منابع موجود در دستگاه کاربر، نیازمندی‌های کاربر، کیفیت شبکه دسترسی و شلوغی سرورهای رایانش لبه‌ای. علاوه بر پارامترهای محیطی، ساختار کاربردهای[3] نرم‌افزاری مورد بررسی نیز در مسئله تاثیر

[1] Jitter

[2] User Equipment

[3] Application

می‌گذارند. برای مثال ممکن است که تمام یا بخشی از یک کاربرد خاص قابلیت تخلیه نداشته باشد.

در پروژه‌ی فعلی **تاخیر سرویس** به عنوان تابع هدف در نظر گرفته شده است. تاخیر همواره یک معیار اصلی در سنجش کیفیت سامانه‌های کامپیوتری بوده است. همچنین با رشد روز افزون صنعت اینترنت اشیاء، کاربردهای جدیدی در سطح شبکه به وجود آمده است که نیازمندی‌های تاخیر بسیار پایین دارند، به طوری که سرورهای رایانش ابری پاسخگوی این نیازمندی نخواهد بود. و از طرفی نیازمندی‌های پردازشی بالا امکان اجرای این کاربردها به صورت محلی و بهنگام[۴] را نمی‌دهد. یک نمونه از این دسته از کاربردها «سامانه مدیریت ترافیک هوشمند» است که نیازمند انجام پردازش‌های سنگین در زمان بسیار کم می‌باشد.

تاخیر سرویس بسته به اجرای محلی و یا تخلیه از مولفه‌های متفاوتی تشکیل می‌شود. در صورت اجرای محلی تاخیر سرویس از موارد زیر تشکیل خواهد بود:

۱. تاخیر انتظار در صف وظیفه d_q

۲. تاخیر اجرا به صورت محلی d_{loc}

و در صورت تخلیه از موارد زیر تشکیل خواهد شد:

۱. تاخیر صف d_q

۲. تاخیر ارسال d_{tx}

۳. تاخیر انتشار $d_{propagation}$

۴. تاخیر اجرا در سرور لبه‌ای d_{server}

۵. تاخیر بازدریافت وظیفه از سرور d_{rx}

[4] Real-time

در پروژه‌ی فعلی روشی برای بدست آوردن استراتژی تخلیه‌ی وظیفه با تاخیر کمینه تحت محدودیت توان مصرفی در محیط رایانش لبه‌ای ارائه خواهیم داد. روش ارائه شده مبتنی بر زنجیره مارکوف گسسته-زمان و برنامه‌ریزی خطی می‌باشد و گسترشی بر روش ارائه شده در [۱] می‌باشد. نوآوری و مزیت اصلی روش پیشنهادی ما نسبت به مقاله ذکر شده قابلیت پشتیبانی از وظایف با نیازمندی‌های پردازشی و شبکه‌ای متفاوت (وظایف ناهمگون) می‌باشد. انگیزه اصلی از این گسترش، تنوع محاسباتی وظایف در محیط‌های اینترنت اشیاء بوده است. به طور مثال در بسیاری از پژوهش‌های حوزه تخلیه‌ی وظیفه در اینترنت اشیاء، وظایف به دو دسته «سبک» و «سنگین» تقسیم می‌شوند. [۳] [۴] برای درک مفهوم وظایف سبک و سنگین می‌توان مثال اتومبیل خودران را در نظر گرفت. در این کاربرد، وظیفه پردازش اطلاعات تصاویر به منظور راندن خودرو یک وظیفه سنگین محسوب می‌شود، در حالی که وظیفه روشن کردن سیستم گرمایشی خودرو بر حسب داده‌ی سنسور دما، یک وظیفه سبک محسوب می‌شود.

ادامه پروژه‌ی فعلی به پنج فصل تقسیم شده است. در فصل ۲ پژوهش‌های مرتبط انجام شده را مرور می‌کنیم. در فصل ۳ به شرح مسئله تخلیه‌ی وظیفه و ساختار رایانش لبه‌ای می‌پردازیم. در فصل ۴ روش پیشنهادی برای بدست آوردن استراتژی تخلیه بهینه را شرح می‌دهیم. در فصل ۵ ابتدا ساختار نرم‌افزاری[5] جدیدی مبتنی بر زبان Kotlin با نام Kompute ارائه می‌دهیم که این امکان را به کاربران و پژوهشگران می‌دهد تا استراتژی تخلیه بهینه را به ازای سیستم دلخواه خود محاسبه کنند و آن استراتژی را با سایر استراتژی‌های پایه[6] مقایسه کنند. در بخش دوم از فصل ۵ با استفاده از Kompute به آزمایش و شبیه‌سازی روش ارائه شده در بخش ۴ می‌پردازیم. در انتها در فصل ۶ یک جمع‌بندی کلی از تمامی مطالب ارائه می‌دهیم و پیشنهاداتی نیز برای گسترش روش پیشنهادی ارائه می‌کنیم.

[5] Framework

[6] Baseline

فصل ۲

مروری بر ادبیات و کارهای انجام شده

پژوهش‌های انجام شده در زمینه تخلیه پردازش[1] را می‌توان بر حسب «ویژگی‌های محیط مسئله» و همینطور «الگوریتم استفاده شده برای حل مسئله» دسته‌بندی کرد. در این فصل ابتدا به معرفی این ویژگی‌ها و الگوریتم‌ها می‌پردازیم و سپس برخی از مقالاتی که ارتباط نزدیکی با پروژه‌ی فعلی دارند را معرفی می‌کنیم.

۲–۱ بررسی مقالات از نظر ویژگی‌های محیط مسئله

در جدول ۲–۱ که برگرفته از [۵] می‌باشد، برخی از ویژگی‌های محیط مسئله و حالت‌های ممکن برای این ویژگی‌ها مشاهده می‌شود.

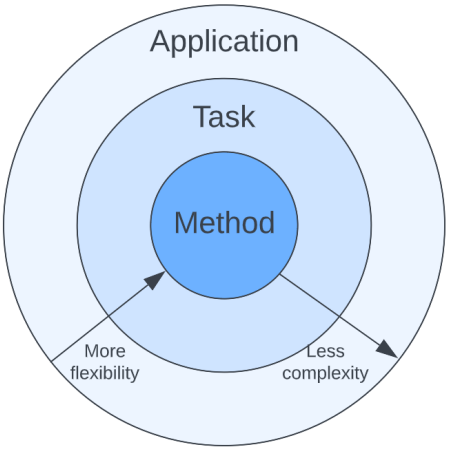
Granularity	User Count	Mobility	Destination	Metric
Application	Single UE	Cloud Server	Single-server	Delay
Task	Multi UE	Edge Server	Multi-server	Energy
Method		Ad hoc		Cost

جدول ۲–۱: تقسیم‌بندی شرایط محیطی مسئله تخلیه پردازش

[1] Computation Offloading

دانه‌بندی (Granularity)

دانه‌بندی به نوع مولفه‌های پردازشی قابل تخلیه در سیستم اشاره دارد. طبق [۵] دانه‌بندی را با سه دسته مختلف (به ترتیب از دانه ریز به دانه درشت) بیان می‌کنیم: کاربرد، وظیفه، متد. هر چه دانه‌بندی ریزتر باشد انعطاف‌پذیری سیستم تخلیه بیشتر خواهد بود، به طوری که به توسعه‌دهندگان نرم‌افزار اجازه خواهد داد تا به طور دقیق مشخص کنند که کدام قسمت‌ها از یک کاربرد خاص تخلیه شوند و کدام قسمت‌ها نشوند. با این حال پیاده‌سازی سیستم‌های تخلیه پردازش به صورت دانه‌ریز به مراتب پیچیده‌تر است. پیاده‌سازی‌های دانه‌ریز همچنین هزینه اضافه[۲] بیشتری برای ساخت محیط‌های مجازی در سرور خواهند داشت. خلاصه‌ای از انواع دانه‌بندی‌ها در شکل ۲-۱ آورده شده است.



شکل ۲-۱: سه دانه‌بندی مختلف در سامانه تخلیه پردازش

به عنوان نمونه در [۶] دانه‌بندی در سطح متد صورت گرفته است، در حالی که در [۱] دانه‌بندی در سطح وظیفه صورت گرفته است. ما نیز در پروژه‌ی فعلی مسئله تخلیه پردازش را در سطح وظیفه حل کرده‌ایم.

[2] Overhead

تعداد کاربران (User Count)

در برخی از مقالات مانند [۱] مسئله تخلیه پردازش تنها برای یک کاربر در نظر گرفته می‌شود در حالیکه در برخی از پژوهش‌ها مانند [۷] از چندین کاربر همزمان نیز پشتیبانی می‌شود. در پروژه‌ی فعلی تعداد کاربران را برای سادگی بیشتر یک در نظر می‌گیریم.

تحرک‌پذیری (Mobility)

به انجام پردازش توسط هر گره[3]ای به جز گره ایجاد کننده وظیفه، تخلیه‌ی وظیفه گفته می‌شود. طبق این تعریف سه نوع از تحرک‌پذیری را متناظر با نوع گره پردازشی می‌توانیم در نظر بگیریم.

۱. پردازش در سرور ابری

۲. پردازش در سرور لبه‌ای

۳. پردازش در شبکه‌ای بدون ساختار[4] (از دستگاه‌های کاربر)

تعداد سرور

مشابه با تعداد کاربران، تعداد سرورهای پردازشی در سامانه تخلیه نیز می‌تواند یک یا بیشتر باشد. برای نمونه در [۷] مسئله تخلیه پردازش برای چندین سرور بررسی شده است. در پروژه‌ی فعلی ما حالت تک سرور را در نظر می‌گیریم.

معیار بهینه‌سازی

معیار بهینه‌سازی به کمیتی اشاره دارد که استراتژی تخلیه پردازش سعی در بهینه‌سازی آن دارد. برخی از معیارهای رایج عبارتند از: تاخیر، انرژی، کیفیت سرویس، و هزینه. برای مثال در [۱] معیار تاخیر، در [۸] معیار انرژی و در [۹] معیار هزینه در نظر گرفته شده است.

[3] Node

[4] Ad-hoc

۲-۲ بررسی مقالات از نظر روش حل مسئله

در جدول ۲-۲ که برگرفته از [۱۰] می‌باشد، یک دسته‌بندی کلی از الگوریتم‌های رایج در حل مسئله تخلیه پردازش مشاهده می‌شود. برای آشنایی بیشتر با این روش ها به [۵] و [۱۰] رجوع شود. در پروژه‌ی فعلی ما از **الگوریتمی** قطعی[5] بر پایه برنامه‌ریزی خطی برای یافتن **استراتژی تخلیه** تصادفی[6] استفاده می‌کنیم.

Model	Examples
Stochastic	Machine learning, Generalized poison distribution, Game theory, Queuing theory, Markov processes, Gaussian processes
Deterministic	Some supervised Machine Learning approaches (e.g., KNN), Linear and non-linear programming, Linear regression equation

جدول ۲-۲: تقسیم‌بندی الگوریتم‌های حل مسئله تخلیه پردازش

۲-۳ پژوهش‌های مرتبط

در [۱] مسئله تخلیه‌ی وظیفه با تاخیر کمینه با استفاده از روشی مبتنی بر زنجیره مارکوف و برنامه‌ریزی خطی حل شده است. در پروژه‌ی فعلی محیط تک کاربر و تک سرور در نظر گرفته شده است. روش ارائه شده در درازمدت عملکرد بهینه دارد اما چندین کاستی دارد از جمله عدم پشتیبانی از وظایف با نیازمندی‌های پردازشی متفاوت و عدم پشتیبانی از موازی‌سازی. پروژه‌ی فعلی گسترشی بر این مقاله است.

در [۱۱] یک مکانزیم تخلیه‌ی وظیفه با هزینه کمینه برای محیط رایانش لبه‌ای متحرک ارائه شده است. محیط در نظر گرفته شده از نظر ثابت بودن طول بازه‌های زمانی و تفاوت وظایف و همچنین

[5] Deteministic

[6] Stochastic

نحوه تعریف مسئله بهینه‌سازی، شبیه به پژوهش ما می‌باشد. اما از نظر معیار و تعداد سرور متفاوت می‌باشد. روش بهینه‌سازی استفاده شده در پروژه‌ی فعلی روش «ضرایب لاگرانژ» می‌باشد که عملکرد سریعی دارد اما لزوما جواب بهینه سراسری را پیدا نمی‌کند و فقط جواب‌های بهینه محلی را پیدا می‌کند.

در [۱۲] و [۱۳] مشابه با پروژه‌ی فعلی ، ناهمگونی وظایف و تقابل[۷] تاخیر و انرژی در نظر گرفته شده است. با این تفاوت که در این دو مقاله از روش بهینه‌سازی لیاپانوف استفاده شده است. همچنین این دو مقاله مسئله تخلیه‌ی وظیفه را در محیط رایانش ابری را در نظر گرفته‌اند و نه رایانش لبه‌ای و همچنین چارچوب نرم‌افزاری ای برای حل مسئله در محیط‌های خاص ارائه نداده اند.

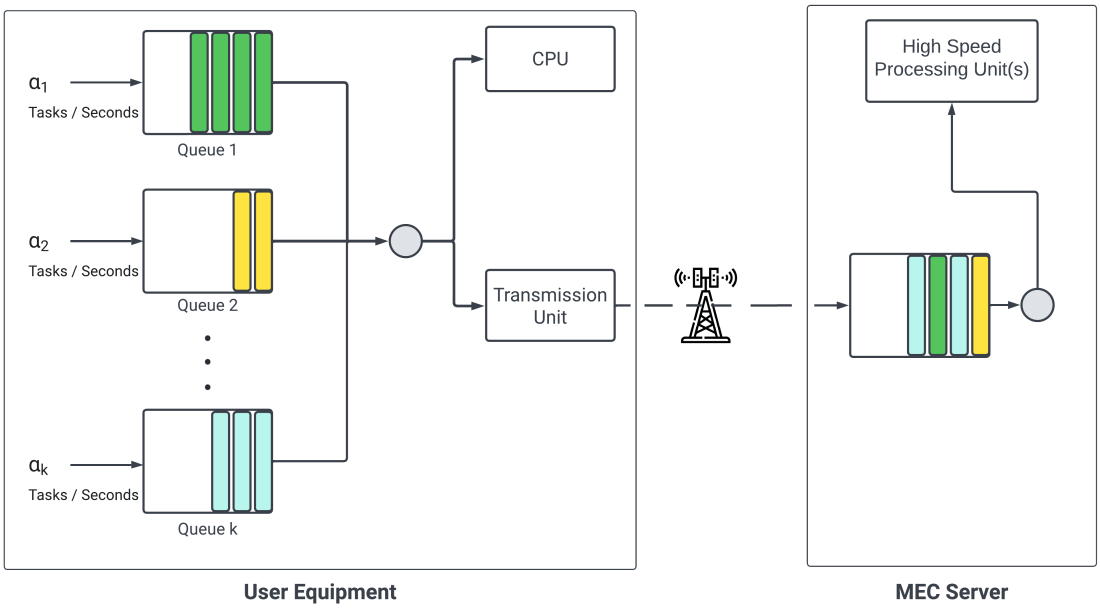
در [۱۴] مسئله تخلیه و زمان‌بندی ارسال و اجرا به صورت همزمان در نظر گرفته شده است و کانال بیسیم به صورت تصادفی مدل شده است و از این ابعاد به پژوهش ما شباهت دارد. معیار بهینه‌سازی در پروژه‌ی فعلی انرژی مصرفی است. روش ارائه شده عملکرد خوبی دارد و به میزان قابل توجی در انرژی مصرفی صرفه‌جویی می‌کند. با این حال مدل در نظر گرفته شده کاستی‌هایی دارد. یک ایراد اصلی فرض وجود تنها یک کاربرد در سیستم است. به عبارت دیگر تاخیر ایجاد شده به واسطه انتظار کاربردها در صف در نظر گرفته نشده است.

[7] Tradeoff

فصل ۳

شرح مسئله

در پروژه‌ی فعلی قصد داریم در یک سامانه رایانش لبه‌ای مطابق با شکل ۳-۱، استراتژی تخلیه‌ای بیابیم که تاخیر سرویس میانگین $\bar{T}$ را تحت محدودیت توان مصرفی P_{max} در درازمدت کمینه کند.



شکل ۳-۱: ساختار کلی سیستم تخلیه پردازش

همانطور که در شکل ۳-۱ مشاهده می‌شود، در سامانه مد نظر سه مولفه اصلی وجود دارد:

۱. دستگاه کاربر (User Equipment)

۲. سرور رایانش لبه‌ای چند-دسترسی (Multi-access Edge Computing Server)

۳. کانال بیسیم

در فصل جاری نحوه عملکرد هر کدام از این مولفه‌ها در قالب مدل‌های تئوری شرح داده می‌شود.

۳-۱ مدل وظایف

فرض می‌شود که k نوع وظیفه مختلف در سیستم رایانش لبه‌ای وجود دارد و به ازای هر نوع وظیفه دقیقا یک صف در سیستم وجود دارد. وظایف نوع i-اُم برای اجرا به صورت محلی[1] احتیاج به L_i بازه زمانی پردازش توسط پردازنده دارند و به منظور تخلیه به سرور رایانش لبه‌ای احتیاج به M_i واحد زمانی ارسال توسط واحد ارسال[2] دارند. همچنین فرض می‌شود که وظایف نوع i-اُم در سرور رایانش لبه‌ای به C_i بازه زمانی پردازش توسط سرور نیاز دارند. برای سادگی بیشتر در ادامه پروژه‌ی فعلی برای اشاره به یک واحد زمانی اجرا توسط پردازنده از عبارت «**قسمت**»[3] استفاده می‌کنیم که انتزاعی از قسمت‌های کد اجرایی است. و برای اشاره به یک واحد زمانی ارسال توسط واحد ارسال از عبارت «**بسته**» استفاده می‌شود.

[1] Local

[2] Transmission Unit

[3] Section

۳-۲ مدل دستگاه کاربر

دستگاه کاربر مطابق با شکل ۳-۱ شامل دو مولفه پردازنده و واحد ارسال می‌باشد. همچنین همانطور که اشاره شد k صف مختلف به ازای هر کدام از انواع وظایف در سیستم وجود دارد. ظرفیت هر صف را برابر با مقدار ثابت Q در نظر می‌گیریم.

در هر بازه زمانی، پردازنده یا به اندازه یک قسمت پردازش انجام می‌دهد و یا بیکار[۴] است. اجرای هر قسمت پردازش توسط پردازنده به میزان P_{loc} وات توان مصرف می‌کند. به طور مشابه واحد ارسال در هر بازه زمانی یا یک بسته را به شبکه ارسال می‌کند یا بیکار است. نکته قابل توجه در مورد واحد ارسال این است که با توجه به شرایط کانال بیسیم، در یک بازه زمانی خاص ممکن است ارسال موفقیت آمیز باشد یا نباشد. فرض می‌شود که ارسال موفقیت آمیز هر بسته به میزان P_{tx} وات توان مصرف می‌کند. توضیحات بیشتر در مورد نحوه کارکرد کانال بی‌سیم در بخش ۳-۴ آورده شده است.

با توجه به توضیحات داده شده می‌توان مدلی برای «حالت دستگاه کاربر»[۵] تعریف کرد. در [۱] برای مشخص کردن حالت دستگاه در زمان t از یک سه تایی مانند $\boldsymbol{\tau}[t] = (q[t], c_T[t], c_L[t])$ استفاده شده است، که در آن $q[t]$ مشخص کننده تعداد وظایف موجود در صف وظایف، $c_T[t]$ مشخص کننده تعداد بسته ارسال شده از وظیفه تخصیص داده شده به واحد ارسال است، و $c_L[t]$ مشخص کننده تعداد قسمت اجرا شده از وظیفه تخصیص داده شده به پردازنده است. همچنین حالت $c_T[t] = 0$ معادل با بیکار بودن واحد ارسال و $c_L[t] = 0$ معادل با بیکار بودن پردازنده تعریف می‌شود. برای مثال سه تایی $(4, 2, 1)$ به این معنی است که ۴ وظیفه در صف وظایف وجود دارد، واحد پردازش در حال تخلیه‌ی وظیفه‌ای است و تا کنون یک بسته از آن وظیفه را ارسال کرده و به عنوان قدم بعدی باید بسته شماره ۲ را ارسال کند. پردازنده نیز در حال اجرای وظیفه‌ای به صورت محلی است و تا کنون یک قسمت از آن وظیفه را اجرا کرده است.

[4] Idle

[5] User Equipment State

با این حال مدل فوق در مسئله تخلیه‌ی وظیفه با چند نوع وظیفه قابل استفاده نیست و نیاز به تغییر دارد. ما در پروژه‌ی فعلی برای تعیین حالت دستگاه کاربر از یک چندتایی[۶] به طول $k+4$ مطابق با رابطه ۳-۱.۲ استفاده می‌کنیم. در این رابطه متغیرهای $q_1[t], \cdots, q_k[t]$ تعداد وظایف موجود از هر نوع وظیفه در صف مربوطه را مشخص می‌کنند. متغیرهای $c_R[t]$ و $c_L[t]$ مشابه با حالت تک صف تعریف می‌شوند و به ترتیب وضعیت واحد ارسال و پردازنده را مشخص می‌کنند. دو متغیر جدید $T_R[t]$ و $T_L[t]$ به ترتیب مشخص کننده نوع وظیفه در حال ارسال توسط واحد ارسال و نوع وظیفه در حال اجرا توسط پردازنده اند.

$$\tau[t] = (q_1[t], q_2[t], \ldots, q_k[t], c_R[t], c_L[t], T_R[t], T_L[t]) \qquad (۳-۱.۲)$$

در پروژه‌ی فعلی به منظور خوانایی بیشتر، چندتایی بیان شده در رابطه ۳-۱.۲ را به صورت زیر نیز نمایش می‌دهیم و این دو صورت معادل هم می‌باشند:

$$\tau[t] = ([q_1[t], q_2[t], \ldots, q_k[t]], c_R[t], c_L[t], T_R[t], T_L[t]) \qquad (۳-۲.۲)$$

رابطه ۳-۳.۲ با تعریف شروط مختلف فضای حالت مسئله را توصیف می‌کند. **(نکته: در رابطه ۳-۳.۲ و سراسر پروژه‌ی فعلی منظور از $\tau\{X\}$ مقدار متغیر X در حالت τ است.)**

$$\begin{aligned}
&\forall \tau \in S, i \in \{1,2,\ldots,k\} \quad 0 \leqslant \tau\{q_i\} \leqslant Q \\
&\forall \tau \in S \quad \tau\{T_L\}, \tau\{T_R\} \in \{0,1,2,\ldots,k\} \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_R\} = 0\} \quad \tau\{C_R\} = 0 \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_R\} \neq 0\} \quad 1 \leqslant \tau\{C_R\} \leqslant M_{\tau\{T_R\}} \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_L\} = 0\} \quad \tau\{C_L\} = 0 \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_L\} \neq 0\} \quad 1 \leqslant \tau\{C_L\} \leqslant L_{\tau\{T_L\}} - 1
\end{aligned} \qquad (۳-۳.۲)$$

[6] Tuple

۳-۳ مدل زمان

وضعیت سیستم تخلیه‌ی وظیفه در فواصل زمانی[۷] با طول ثابت Δ میلی ثانیه بررسی می‌شود. برای مثال حالت دستگاه کاربر را در بازه زمانی t-اُم با $\tau[t]$ مشخص می‌کنیم، و حالت دستگاه در بازه زمانی $t+1$ را با $\tau[t+1]$ مشخص می‌کنیم و فاصله بین این دو بازه زمانی Δ میلی ثانیه است.

بررسی زمان به صورت واحدهای گسسته به منظور ساده‌سازی مسئله و همچنین گسترش پذیری آن به شرایط محیطی مختلف صورت گرفته است. در عمل، یک مقدار قابل استفاده برای Δ طول بازه‌های زمانی شبکه دسترسی[۸] مورد نظر است. برای مثال در شبکه‌های LTE طول هر بازه زمانی ۰/۵ میلی‌ثانیه می‌باشد. [۱۵]

۳-۴ مدل کانال بیسیم

در پروژه‌ی فعلی مشابه با [۱] کانال بی‌سیم را به صورت تصادفی مدل می‌کنیم[۹] یکی از دلایل اصلی برای مدل‌سازی کانال به صورت تصادفی، وجود نویز و ناپایداری در ارتباطات بیسیم است. کانال بی‌سیم را با یک مدل ساده احتمالی دوجمله‌ای مدل می‌کنیم به این صورت که ارسال هر بسته توسط واحد ارسال با احتمال β موفقیت آمیز خواهد بود و با احتمال $1-\beta$ ناموفق خواهد بود. در عمل مقدار β با توجه به رابطه ۳-۴.۴ (رابطه شنون) محاسبه می‌شود، که در آن R مشخص کننده سایز هر بسته است، $r(t)$ مشخص کننده نرخ ارسال در زمان t ، B پهنای باند سیستم، $\gamma[t]$ مقدار بهره کانال[۱۰] و N_0 مشخص کننده اندازه نویز کانال است.

$$\begin{aligned} &\beta = P(r(t) \geq R) \\ &r(t) = B \log_r \left(1 + \frac{\gamma[t] P_{\text{tx}}}{N_0 B}\right) \end{aligned} \qquad (۳-۴.۴)$$

[7] Time Slot

[8] Access Network

[9] Stochastic Channel

[10] Channel Gain

۳-۵ مفهوم کنش

یک استراتژی تخلیه در هر بازه زمانی مانند t می‌بایست یک کنش[11] مانند $v[t]$ را برای اجرا توسط دستگاه کاربر انتخاب کند. اجرای هر کنش می‌تواند حالت دستگاه کاربر را تغییر دهد. برای درک بهتر مفهوم کنش، ابتدا مشابه [۱] حالتی را در نظر می‌گیریم که تنها یک صف (یک نوع وظیفه) در سیستم وجود داشته باشد. در این حالت می‌توانیم مجموعه کنش‌ها را با چهار عضو مطابق جدول ۳-۱ مشخص کنیم.

ID	Transmit	Local Execution	Description
1	False	False	No operation
2	False	True	Add to CPU
3	True	False	Add to TU
4	True	True	Add to both units

جدول ۳-۱: لیست کنش‌ها در سیستم با یک صف وظیفه

به طور مشابه در شرایطی که بیش از یک نوع وظیفه در سیستم وجود داشته باشد مجموعه کنش‌های ممکن مطابق با جدول ۳-۲ بدست می‌آید.

ID	Transmit	Local Execution	Description	Count
$\{1\}$	False	False	No operation	1
$\{2, ..., k+1\}$	False	True	Add to CPU	k
$\{k+2, ..., 2k+1\}$	True	False	Add to TU	k
$\{2k+2, ..., 2k+k*k-1\}$	True	True	Add to both units	k^2

جدول ۳-۲: دسته‌بندی کنش‌ها در سیستم با k صف

اجرای هر کنش طبعا ممکن است که حالت سیستم را تغییر دهد. به طور مثال اجرای هر کنش نوع Add To CPU یک وظیفه را از صف مربوطه بر می‌دارد، بنابراین طول صف مطابق $q_i[t+1] = q_i[t] - 1$ تغییر می‌کند. با اجرای این کنش همچنین وضعیت پردازنده از $c_L[t] = 0$ یعنی حالت بیکار به $c_L[t+1] = 1$ تغییر خواهد کرد زیرا قسمت اول وظیفه مربوطه در بازه زمانی t انجام خواهد شد. به

[11] Action

طور مشابه برای سایر کنش‌ها نیز میتوان توابع انتقال[12] مشخص تعریف کرد که با گرفتن یک حالت ورودی، حالت خروجی را محاسبه نماید. به دلیل پیچیدگی روابط این توابع، از توضیح بیشتر در این بخش صرف نظر شده است. برای مشاهده منطق دقیق این توابع در قالب کد، به پیوست ۱ مراجعه شود.

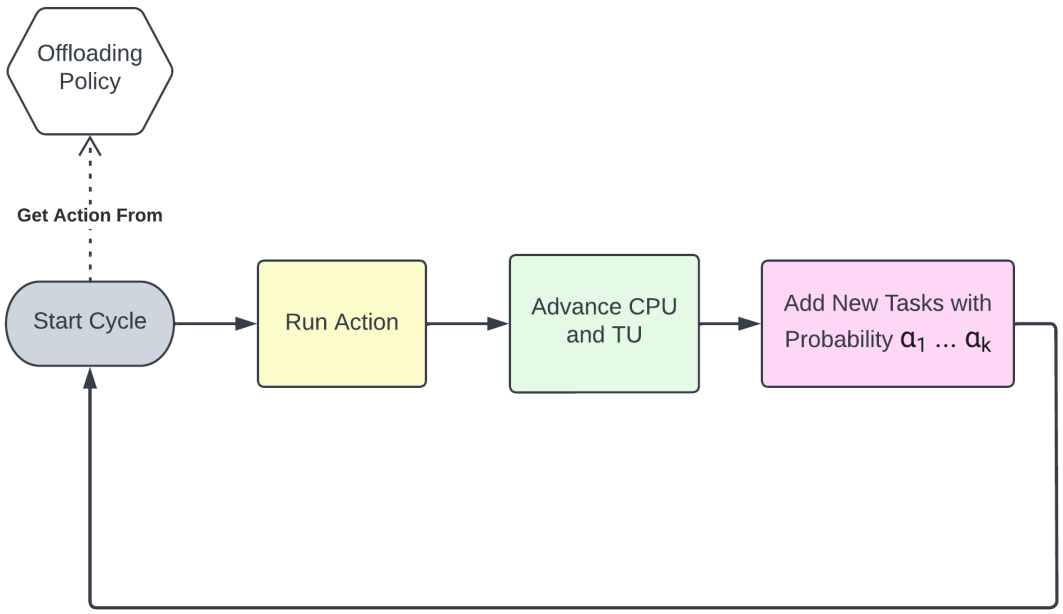
۳-۶ استراتژی تخلیه

استراتژی تخلیه در هر بازه زمانی تصمیم می‌گیرد که دستگاه کاربر چه کنشی را اجرا کند. بنابراین استراتژی تخلیه یک تابع مانند $G(\tau)$ می‌باشد که با گرفتن حالت دستگاه کاربر $\tau[t]$ به عنوان ورودی، یک کنش مانند a را به عنوان خروجی می‌دهد. لازم به ذکر است که در اینجا این تابع را به صورت مفهومی انتزاعی در نظر می‌گیریم و در فصل‌های آتی به طور دقیق به نحوه بدست آوردن تابع بهینه $g(\tau)^*$ خواهیم پرداخت.

[12] Transition Function

۳-۷ روند فعالیت سیستم تخلیه‌ی وظیفه

نحوه عملکرد دستگاه کاربر در هر بازه زمانی مطابق با فرآیند مشخص شده در شکل ۳-۲ می‌باشد. در هر باز، دستگاه کاربر ابتدا کنش اجرایی را از یک استراتژی تخلیه دریافت می‌کند. سپس کنش انتخاب شده توسط دستگاه کاربر اجرا خواهد شد که ممکن است منجر به تغییر حالت دستگاه شود. سپس پردازنده و واحد ارسال هر کدام در صورت فعال بودن به اندازه یک بازه زمانی فعالیت خواهند کرد. در انتها وظایف جدید با احتمالات $\alpha_1, \cdots, \alpha_k$ به صف‌های وظایف اضافه خواهند شد.



شکل ۳-۲: روند فعالیت دستگاه کاربر

فصل ۴

روش پیشنهادی

در این فصل الگوریتمی ارائه خواهیم داد که با استفاده از آن می‌توان مسئله یافتن استراتژی تخلیه با تاخیر کمینه را که در فصل قبل تشریح شد را حل کرد. استراتژی خروجی توسط الگوریتم از نوع تصادفی می‌باشد و برای بدست آوردن آن از مفاهیمی مانند زنجیره مارکوف و برنامه‌ریزی خطی استفاده خواهد شد.

۴–۱ استراتژی تخلیه تصادفی

با استفاده از مدل‌های توصیف شده در فصل قبل می‌توانیم یک تعریف ریاضی از «استراتژی تخلیه تصادفی» داشته باشیم. مشابه با مقاله [۱] استراتژی تخلیه تصادفی را به صورت یک توزیع احتمالی مانند g_τ^a بر روی مجموعه $S \times A$ تعریف می‌کنیم. در اینجا عبارت $S \times A$ نمایانگر ضرب دکارتی مجموعه تمام حالت‌های سیستم در مجموعه تمام کنش‌های ممکن در سیستم است. یک نکته قابل توجه این است که برخی از دو تایی‌های حاصل از این ضرب دکارتی هیچ گاه در واقعیت امکان‌پذیر نیست. برای مثال در حالتی که صف خالی باشد تنها یک کنش امکان پذیر است و آن هم کنش شماره ۱ (No Operation) است. با این حال برای سادگی در توضیح تئوری روش حل مسئله، این دو تایی‌ها را نیز در دامنه تابع توزیع احتمالی استراتژی تخلیه در نظر می‌گیریم تا همواره تعداد اعضای دامنه تابع احتمال برابر با $|S| \cdot |A|$ باشد.

همچنین طبق تعریف توزیع احتمال، رابطه ۴-۱.۱ باید برای هر استراتژی تخلیه تصادفی برقرار باشد.

$$\sum_{\tau \in S} \sum_{a \in A} g_{\tau}^{a} = 1 \qquad (۴-۱.۱)$$

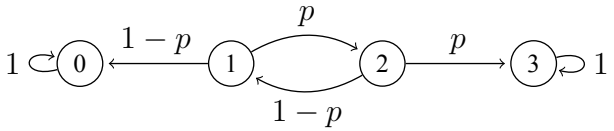
۴-۲ مدل زنجیره مارکوف دستگاه کاربر

در این قسمت ابتدا مدل آماری زنجیره مارکوف گسسته-زمان را معرفی می‌کنیم و سپس توضیح می‌دهیم که چگونه می‌توان با استفاده از این مدل معیارهای تاخیر و توان مصرفی میانگین را برای یک سیستم تخلیه‌ی وظیفه محاسبه کرد.

تعریف ۴.۱. دنباله‌ای از متغیرهای تصادفی $X_1, X_2, \ldots$ را که احتمال تغییر وضعیت از زمان t به $t+1$ مستقل از وضعیت‌های قبلی باشد را یک **زنجیره مارکوف گسسته-زمان** می‌نامند. این گزاره را به بیان متغیرهای تصادفی و تابع احتمال به صورت رابطه زیر نشان می‌دهیم.

$$\Pr\left(X_{t+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_t\right) = \Pr\left(X_{t+1} = x \mid X_t = x_t\right)$$

زنجیره مارکوف گسسته-زمان را می‌توان با گراف جهت‌دار نیز نمایش داد. در شکل ۴-۱ یک زنجیره نمونه مشاهده می‌شود.



شکل ۴-۱: یک زنجیره مارکوف نمونه برای مسئله پاکباختگی قمارباز[1]

[1] The Gambler's ruin

تعریف ۲.۴. زنجیره مارکوف گسسته زمان $X(t)$ را **همگن-زمان** می‌گوییم اگر شرط زیر همواره برقرار باشد:

$$P\left(X_{n+1}=j \mid X_n=i\right)=P\left(X_1=j \mid X_0=i\right)$$

به عبارت دیگر یعنی احتمالات مربوط به انتقال بین حالت‌ها به زمان t وابسته نیستند. در این حالت احتمال انتقال زنجیره از حالت i به j را با عبارت $p_{ij}=P\left(X_1=j \mid X_0=i\right)$ نمایش می‌دهیم و همچنین ماتریس انتقال را با $P=(p_{ij})$ نمایش می‌دهیم.

طبق تعاریف ۱.۴ و ۲.۴ می‌توان حالت دستگاه کاربر در طی زمان را به صورت یک زنجیره مارکوف گسسته‌زمان در نظر گرفت به طوری که $\tau[t]$ حالت زنجیره در زمان t را مشخص می‌کند. همچنین ماتریس انتقال χ را اینگونه تعریف می‌کنیم که $\chi_{\tau,\tau'}$ احتمال انتقال از حالت τ به τ' را مشخص می‌کند. این ماتریس انتقال به ازای یک استراتژی تخلیه داده شده و پارامترهای سیستمی مشخص قابل محاسبه می‌باشد. به طور دقیق‌تر احتمال انتقال از حالتی مانند τ به τ' بستگی به مقادیر زیر دارد:

- استراتژی تخلیه g_τ^a
- احتمال ورود وظایف $\alpha_1, \cdots, \alpha_k$
- احتمال موفقیت واحد ارسال β

برای نمونه در سیستم تخلیه‌ی وظیفه‌ای با ویژگی‌های مشخص شده در جدول ۴-۱ احتمال انتقال به حالات بعدی مطابق با جدول ۴-۲ می‌باشد. برای سادگی، در اینجا از توضیح بیشتر در مورد نحوه محاسبه مقادیر درایه‌های ماتریس انتقال صرف نظر می‌کنیم. برای آگاهی از نحوه محاسبه این مقادیر در قالب کد به پیوست ۳ رجوع شود.

Parameter	M_1	M_2	L_1	L_2	C_1	C_2	β	P_{tx}	P_{loc}	P_{max}	t_{rx}
Value	1	3	7	2	1	1	0.95	1	0.8	1.6	0

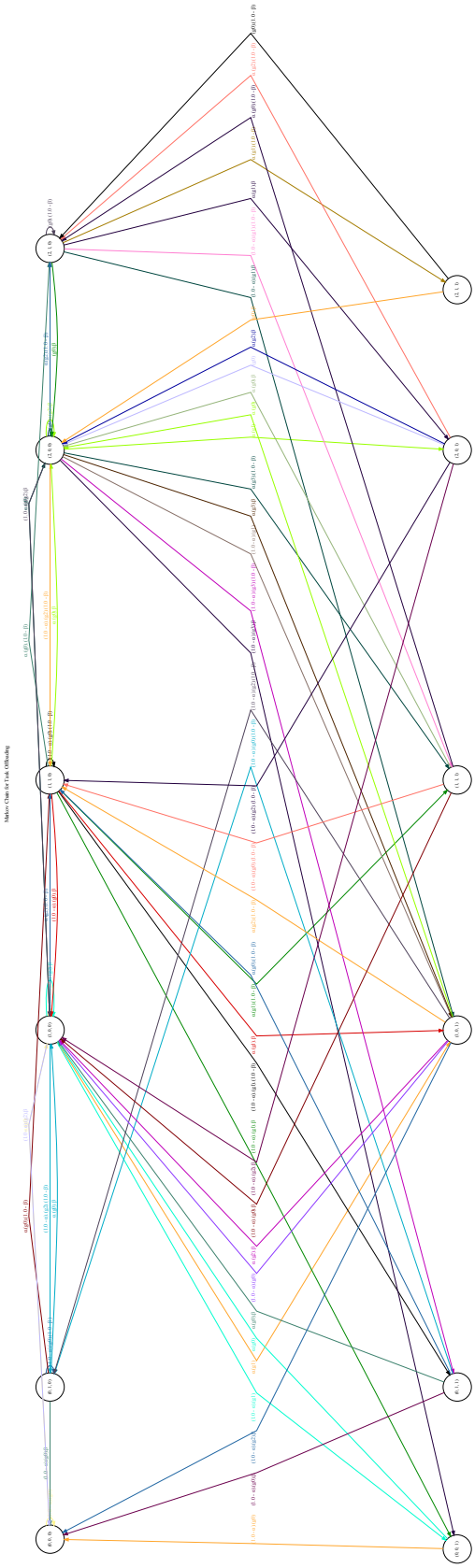
جدول ۴-۱: پارامترهای محیط رایانش لبه‌ای در سناریو دو صف با یک صف ثابت

τ'	$\chi_{\tau,\tau'}$
$([1,1],0,2,0,1)$	$g_\tau^{NoOperation} * (1-\alpha_1) * (1-\alpha_2)$
$([2,1],0,2,0,1)$	$g_\tau^{NoOperation} * \alpha_1 * (1-\alpha_2)$
$([1,2],0,2,0,1)$	$g_\tau^{NoOperation} * (1-\alpha_1) * \alpha_2$
$([2,2],0,2,0,1)$	$g_\tau^{NoOperation} * \alpha_1 * \alpha_2$
$([0,1],0,2,0,1)$	$g_\tau^{AddToTU(1)} * \beta * (1-\alpha_1) * (1-\alpha_2)$
$([0,1],1,2,1,1)$	$g_\tau^{AddToTU(1)} * (1-\beta) * (1-\alpha_1) * (1-\alpha_2)$
$([1,1],0,2,0,1)$	$g_\tau^{AddToTU(1)} * \beta * \alpha_1 * (1-\alpha_2)$
$([1,1],1,2,1,1)$	$g_\tau^{AddToTU(1)} * (1-\beta) * \alpha_1 * (1-\alpha_2)$
$([0,2],0,2,0,1)$	$g_\tau^{AddToTU(1)} * \beta * (1-\alpha_1) * \alpha_2$
$([0,2],1,2,1,1)$	$g_\tau^{AddToTU(1)} * (1-\beta) * (1-\alpha_1) * \alpha_2$
$([1,2],0,2,0,1)$	$g_\tau^{AddToTU(1)} * \beta * \alpha_1 * \alpha_2$
$([1,2],1,2,1,1)$	$g_\tau^{AddToTU(1)} * (1-\beta) * \alpha_1 * \alpha_2$
$([1,0],2,2,2,1)$	$g_\tau^{AddToTU(2)} * \beta * (1-\alpha_1) * (1-\alpha_2)$
$([1,0],1,2,2,1)$	$g_\tau^{AddToTU(2)} * (1-\beta) * (1-\alpha_1) * (1-\alpha_2)$
$([2,0],2,2,2,1)$	$g_\tau^{AddToTU(2)} * \beta * \alpha_1 * (1-\alpha_2)$
$([2,0],1,2,2,1)$	$g_\tau^{AddToTU(2)} * (1-\beta) * \alpha_1 * (1-\alpha_2)$
$([1,1],2,2,2,1)$	$g_\tau^{AddToTU(2)} * \beta * (1-\alpha_1) * \alpha_2$
$([1,1],1,2,2,1)$	$g_\tau^{AddToTU(2)} * (1-\beta) * (1-\alpha_1) * \alpha_2$
$([2,1],2,2,2,1)$	$g_\tau^{AddToTU(2)} * \beta * \alpha_1 * \alpha_2$
$([2,1],1,2,2,1)$	$g_\tau^{AddToTU(2)} * (1-\beta) * \alpha_1 * \alpha_2$

جدول ۴-۲: مقادیر ماتریس انتقال $\chi_{\tau,\tau'}$ در صورت حضور در حالت $\tau = ([1,1],0,1,0,1)$

زنجیره مارکوف را می‌توان به صورت گراف جهت‌دار نیز توصیف کرد به طوری که درایه $p_{i,j}$ در ماتریس انتقال معادل یک یال جهت‌دار از راس i به راس j با وزن $p_{i,j}$ می‌باشد. بنابراین می‌توان گفت که جدول ۴-۲ یال‌های گراف با راس مبدا $\tau = ([1,1],0,1,0,1)$ را مشخص می‌کند. در شکل ۴-۲ گراف جهت‌دار متناظر با زنجیره مارکوف سیستم نمونه‌ای رسم شده است. در این سیستم یک صف وظیفه وجود دارد، $Q = 2$ و هر وظیفه دو قسمت و یک بسته دارد.[۲] با توجه به اینکه تنها یک نوع وظیفه در سیستم وجود دارد از متغیرهای T_L و T_R در فضای حالت صرف نظر شده است. سه‌تایی (x,y,z) را بیانگر حالتی در نظر می‌گیریم که در آن x وظیفه در صف وجود دارد، واحد ارسال در وضعیت y قرار دارد و پردازنده z قسمت از وظیفه تخصیص داده شده را اجرا کرده است.

[۲] کد استفاده شده برای رسم این گراف در آدرس https://github.com/dalisyron/OffloadingVisualizer موجود می‌باشد



شکل ۴-۲: زنجیره مارکوف سیستم تخلیه در قالب گراف جهت دار (برای مشاهده جزئیات زوم کنید)

۴-۳ محاسبه تاخیر و توان میانگین با کمک توزیع پایدار

به منظور محاسبه معیارهای توان مصرفی میانگین و تاخیر سرویس میانگین لازم است که بتوانیم درباره وضعیت سیستم تخلیه‌ی وظیفه در طولانی‌مدت استنتاج کنیم. در همین راستا مفهوم **توزیع پایدار** را برای زنجیره مارکوف تعریف می‌کنیم.

تعریف ۳.۴. توزیع احتمالی مانند p_i را یک **توزیع پایدار** برای زنجیره مارکوف با ماتریس انتقال P می‌گوییم هر گاه شرط زیر در آن برقرار باشد:

$$\pi = \pi P \quad \iff \quad \pi_j = \sum_i \pi_i P_{ij} \quad \forall j.$$

یک سوالی که ممکن است بوجود بیاید این است که آیا هر زنجیره مارکوف گسسته‌زمانی توزیع پایدار دارد؟ برای پاسخ به این سوال لازم است دو مفهوم زنجیره مارکوف تقلیل‌ناپذیر و غیرمتناوب را تعریف کنیم.

تعریف ۴.۴. اگر رسیدن از هر نقطه به نقطه دیگر از فضای حالت با احتمال مثبت در زنجیره مارکوف میسر باشد، زنجیره را **تقلیل‌ناپذیر** گویند. به بیان ریاضی می‌توان تقلیل‌ناپذیر بودن زنجیره مارکوف را به صورت زیر نشان داد.

$$\Pr\left(X_{n_{ij}} = j \mid X_0 = i\right) = p_{ij}^{(n_{ij})} > 0$$

تعریف ۵.۴. تناوب $d(i)$ برای حالت i به صورت $d(i) = \gcd\{n : P_{ii}^n > 0\}$ تعریف می‌شود، که به معنی بزرگ‌ترین مقسوم علیه مشترک تعداد مراحل ممکن است به صورتی که از i شروع کرده و به i برگردیم. یک زنجیره مارکوف تقلیل‌ناپذیر را متناوب با تناوب d می‌گوییم اگر تمامی حالت‌ها تناوبی برابر با $d > 1$ را داشته باشند. یک زنجیره مارکوف تقلیل ناپذیر را **غیرمتناوب** می‌گوییم اگر تمامی حالت‌ها تناوب برابر با ۱ داشته باشند.

قضیه ۱.۴. (همگرایی) هر زنجیره مارکوف تقلیل‌ناپذیر و غیر متناوب دارای یک توزیع پایدار منحصر به فرد مانند π می‌باشد.

حال با استفاده از قضیه ۱.۴ ثابت می‌کنیم که زنجیره مارکوف سیستم تخلیه‌ی وظیفه دارای توزیع پایدار منحصر به فرد است. برای سادگی فرض می‌کنیم که سامانه یک صف دارد و سپس نحوه بسط نتیجه به چندین صف را توضیح می‌دهیم.

قضیه ۲.۴. زنجیره مارکوف مربوط به سیستم تخلیه تک صف تقلیل ناپذیر است.

اثبات:

قسمت الف) با توجه به تعریف سیستم تخلیه می‌دانیم که از هر حالت غیر شروع مانند $(0,0,0) \neq (x,y,z)$ می‌توان به حالت شروع رفت. به این منظور کافی است که تمام وظایف داخل صف به نحوی (اجرا یا ارسال) به اتمام برسند و وظیفه جدیدی نیز در این حین وارد سیستم نشود.

قسمت ب) همچنین می‌توان ثابت کرد که از حالت شروع $(0,0,0)$ می‌توان به هر حالت دیگر (x,y,z) رفت. به این منظور دنباله رخدادهای زیر را در نظر بگیرید:

۱. ورود x وظیفه جدید

۲. انتقال یک وظیفه به واحد ارسال و ورود یک وظیفه جدید، هر دو در صورتی که $y > 0$

۳. پیشرفت واحد ارسال به مدت y سیکل و عدم ورود وظیفه جدید در این حین

۴. انتقال یک وظیفه به پردازنده و ورود یک وظیفه جدید، هر دو در صورتی که $z > 0$

۵. پیشرفت واحد ارسال به مدت z سیکل و عدم ورود وظیفه جدید در این حین

با توجه به نتایج بخش الف و ب می‌توان نتیجه گرفت که از گشتی با احتمال مثبت از هر حالت به حالت دیگر وجود دارد بنابراین طبق تعریف زنجیره تقلیل‌ناپذیر است.

قضیه ۳.۴. زنجیره مارکوف مربوط به سیستم تخلیه تک صف غیر متناوب است.

اثبات:

به منظور اثبات این قضیه فقط کافی است که به این نکته توجه کنیم که حالت $(0,0,0)$ دارای تناوب یک می‌باشد زیرا با احتمالی مثبت (متناظر با رخداد عدم ورود وظیفه و کنش No Operation) می‌توان در همان حالت ماند. با توجه به همین نکته و تقلیل‌ناپذیر بودن زنجیره میتوانیم نتیجه بگیریم که سایر حالت‌ها نیز باید تناوب یک داشته باشند. بنابراین زنجیره غیرمتناوب است.

با توجه به قضایای ۲.۴ و ۳.۴ و قضیه همگرایی می‌توان نتیجه گرفت که زنجیره مارکوف سیستم تخلیه تک صف دارای توزیع پایدار منحصر به فرد می مطابق با رابطه ۴-۲.۳ می‌باشد. برای بسط این اثبات به حالت چند صف اثبات غیرمتناوب بودن یکسان خواهد بود و در اثبات تقلیل‌ناپذیر بودن، رخداد اول به ورود $x_1, \cdots, x_k$ وظیفه از انواع مختلف تغییر پیدا می‌کند.

$$\begin{cases} \sum_{\boldsymbol{\tau}' \in \mathcal{S}} \chi_{\boldsymbol{\tau}',\boldsymbol{\tau}} \pi_{\boldsymbol{\tau}'} = \pi_{\boldsymbol{\tau}}, \forall \boldsymbol{\tau} \in \mathcal{S} \\ \sum_{\boldsymbol{\tau} \in \mathcal{S}} \pi_{\boldsymbol{\tau}} = 1 \end{cases} \quad (۲.۳-۴)$$

۴-۴ محاسبه تاخیر میانگین

تاخیر هر وظیفه شامل تاخیر انتظار در صف وظایف و تاخیر پردازش می‌باشد. به منظور بدست آوردن تاخیر میانگین سیستم ابتدا θ_i را به عنوان کسری از وظایف سیستم در طولانی مدت که از نوع i هستند تعریف می‌کنیم. اگر طول صف‌ها به مقدار کافی بزرگ باشد و همچنین استراتژی تخلیه‌ای داشته باشیم که منجر به پر شدن صف و اتلاف وظیفه[3] نشود مقدار θ_i طبق رابطه ۴-۳.۴ بدست می‌آید.

$$\theta_i = \frac{\alpha_i}{\sum_{j=1}^{k} \alpha_j} \quad (۳.۴-۴)$$

[3] Task Loss

پارامتر t_q^i را برابر با مقدار میانگین تاخیر انتظار در صف مربوط به وظایف نوع i تعریف می‌کنیم. طبق قانون Little می‌توان مقدار این تاخیر را بر اساس رابطه ۴-۴.۴ بدست آورد. همانطور که پیش‌تر ذکر شد برای برقراری این رابطه لازم است که اتلاف وظیفه در صف هیچ‌گاه رخ ندهد. به عبارت دیگر با فرض اینکه استراتژی تخلیه‌ی ارائه شده «کارامد» باشد این رابطه برقرار است. در پیاده‌سازی عملی، محدودیت «کارآمد» بودن یک استراتژی بدین گونه تعریف شده است که احتمال پر بودن صف حداکثر مقداری ناچیز باشد.

$$t_q^i = \frac{\theta_i}{\alpha_i} \sum_{j=0}^{Q} i \cdot \Pr\{q_i[t] = i\} = \frac{1}{\alpha} \sum_{\tau \in S} \tau\{q_i\} \cdot \pi_\tau \qquad (۴.۴-۴)$$

همچنین t_{tx}^i را به عنوان تاخیر ارسال میانگین یک وظیفه از نوع i توسط واحد ارسال تعریف می‌کنیم که مقدار آن بر اساس امید ریاضی موفقیت در فرآیند برنولی مطابق با رابطه ۵-۴.۴ بدست می‌آید.

$$t_{tx}^i = M_i \sum_{j=1}^{\infty} j(1-\beta)^{(j-1)}\beta \qquad (۵.۴-۴)$$

به یاد داریم که مقدار تاخیر در صورت پردازش محلی برای وظایف نوع i برابر L_i می‌باشد. تاخیر اجرا در صورت تخلیه‌ی وظیفه به صورت مجموع زمان ارسال وظیفه t_{tx}^i زمان اجرا در سرور لبه‌ای C_i و تاخیر دریافت نتیجه از سرور t_{rx}^i محاسبه می‌شود.

$$t_c^i = t_{tx}^i + C_i + t_{rx} \qquad (۶.۴-۴)$$

در نتیجه می‌توان تاخیر اجرای میانگین وظایف نوع i را نیز مطابق رابطه ۷-۴.۴ بیان کرد.

$$t_p^i = \eta_i L_i + (1-\eta_i) t_c^i \qquad (۷.۴-۴)$$

که در آن η_i بیانگر کسری از وظایف نوع i می‌باشد که در طولانی‌مدت به صورت محلی اجرا می‌شوند

و مطابق با رابطه ۴-۸.۴ بدست می‌آید.

$$\eta_i = \frac{\sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_3^i\cup\mathcal{S}_5^i} \pi_{\boldsymbol{\tau}} g_{\boldsymbol{\tau}}^a}{\sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_2^i\cup\mathcal{S}_3^i\cup\mathcal{S}_4^i} \pi_{\boldsymbol{\tau}} g_{\boldsymbol{\tau}}^a + 2\sum_{\boldsymbol{\tau},\boldsymbol{a}\in S_5^i} \pi_{\boldsymbol{\tau}} g_{\boldsymbol{\tau}}^a} \tag{۴-۴.۸}$$

که در آن $S_1^i, \cdots, S_5^i$ به صورت زیر تعریف می‌شوند:

(۴-۴.۹)

$$S_1^i = \{\boldsymbol{\tau}, \boldsymbol{a} \in \mathcal{S} \times A | type(a) = AddToCPU \wedge cpuQueue(a) = i\}$$

$$S_2^i = \{\boldsymbol{\tau}, \boldsymbol{a} \in \mathcal{S} \times A | type(a) = AddToTU \wedge tuQueue(a) = i\}$$

$$S_3^i = \{\boldsymbol{\tau}, \boldsymbol{a} \in \mathcal{S} \times A | type(a) = AddToBoth \wedge cpuQueue(a) = i \wedge tuQueue(a) \neq i\}$$

$$S_4^i = \{\boldsymbol{\tau}, \boldsymbol{a} \in \mathcal{S} \times A | type(a) = AddToBoth \wedge cpuQueue(a) \neq i \wedge tuQueue(a) = i\}$$

$$S_5^i = \{\boldsymbol{\tau}, \boldsymbol{a} \in \mathcal{S} \times A | type(a) = AddToBoth \wedge cpuQueue(a) = i \wedge tuQueue(a) = i\}$$

در رابطه فوق تابع $type(a)$ نوع کنش را مشخص می‌کند و یکی از چهار نوع بیان شده در بخش ۳-۵ می‌باشد. توابع $cpuQueue(a)$ و $tuQueue(a)$ نیز نوع وظیفه مربوط به کنش a را مشخص می‌کنند.

با استفاده از روابط بالا همچنین می‌توانیم میانگین تاخیر سرویس هر وظیفه در سیستم را طبق رابطه ۴-۱۰.۴ محاسبه کنیم. رابطه بدست آمده برای $\bar{T}$ همچنین مشخص کننده تابع هدف در مسئله پیدا کردن استراتژی تخلیه بهینه می‌باشد.

$$\bar{T} = \sum_{i=1}^{k} \theta_i \left(t_q^i + t_p^i\right) \tag{۴-۴.۱۰}$$

۴-۵ توان مصرفی میانگین

اگر پارامتر μ_τ^{loc} و μ_τ^{tx} را به ترتیب به عنوان احتمال فعالیت پردازنده در حالت τ و احتمال وجود درخواست ارسال وظیفه در حالت τ تعریف کنیم، و v_{loc} و v_{tx} را به صورت مجموع این دو پارامتر در فضای حالت مسئله تعریف کنیم، آنگاه توان مصرفی میانگین طبق رابطه زیر بدست می‌آید:

$$\begin{aligned}\bar{P} &= \sum_{\boldsymbol{\tau}\in\mathcal{S}} \pi_{\boldsymbol{\tau}} \left(\mu_{\boldsymbol{\tau}}^{loc} P_{loc} + \beta \mu_{\boldsymbol{\tau}}^{tx} P_{tx}\right) \\ &= \sum_{\boldsymbol{\tau}\in\mathcal{S}} \pi_{\boldsymbol{\tau}} \left(\mu_{\boldsymbol{\tau}}^{loc} P_{loc}\right) + \sum_{\boldsymbol{\tau}\in\mathcal{S}} \pi_{\boldsymbol{\tau}} \left(\beta \mu_{\boldsymbol{\tau}}^{tx} P_{tx}\right) \\ &= v_{loc} P_{loc} + \beta v_{tx} P_{tx}\end{aligned} \tag{۱۱.۵-۴}$$

در اینجا فرض می‌کنیم که مقادیر μ_τ^{loc} و μ_τ^{tx} از قبل معلوم است. در پیوست ۲ نحوه بدست آوردن مقادیر v_{tx} و v_{loc} در قالب کد شرح داده شده است.

۴-۶ استراتژی تخلیه‌ی وظیفه بهینه

با توجه به توابع بدست آمده برای تاخیر و توان مصرفی میانگین در بخش‌های پیشین، حال می‌توانیم مسئله پیدا کردن استراتژی تخلیه بهینه را به صورت یک مسئله بهینه سازی مانند $\mathcal{P}_1$ بیان کنیم:

$$\mathcal{P}_1 : \min_{\{g_\tau^a\}} \bar{T} = \left(\sum_{i=1}^{k} \frac{1}{\alpha_i} \sum_{\tau\in S} \tau\{q_i\} \cdot \pi_\tau\right) + T_p^0$$

$$\text{s.t.} \begin{cases} \bar{P} \le \bar{P}_{\max} \\ \sum_{\boldsymbol{\tau}'\in\mathcal{S}} \chi_{\boldsymbol{\tau}',\boldsymbol{\tau}} \pi_{\boldsymbol{\tau}'} = \pi_{\boldsymbol{\tau}}, \boldsymbol{\tau} \in \mathcal{S}, \\ \sum_{\boldsymbol{\tau}\in S} \pi_{\boldsymbol{\tau}} = 1, \\ \sum_{\alpha\in A} g_\tau^\alpha = 1, \forall \tau \in S \\ g_\tau^a \ge 0, \forall \tau \in S,\ a \in A \end{cases} \tag{۱۲.۶-۴}$$

که در آن T_p^0 برابر با تاخیر اجرای میانگین است که به ازای مقادیر داده شده از $\eta_0, \cdots, \eta_k$ مقداری ثابت دارد و از رابطه زیر بدست می‌آید:

$$T_p^0 = \sum_{i=1}^{k} (\eta_i L_i + (1-\eta_i) t_c^i) \tag{۱۳.۶-۴}$$

مسئله $\mathcal{P}_1$ به دلیل وجود پارامتر η_i در تابع هدف یک مسئله خطی نیست. با این حال می‌توانیم با استفاده از تغییری کوچک مسئله را به مجموعه‌ای از مسائل برنامه‌ریزی خطی تبدیل کنیم. به این منظور مشابه با [۱] ابتدا از تعریف «معیار احاطه[۴]» در زنجیره مارکوف استفاده می کنیم. به این منظور مجموعه متغیرهای جایگزین $\{x_\tau^a\}$ را طبق رابطه $x_\tau^a = \pi_\tau g_\tau^a$ تعریف می‌کنیم. به عبارتی x_τ^a برابر با احتمال حضور در حالت τ و انتخاب کنش a می‌باشد. همچنین طبق تعریف می‌دانیم که $\sum_{a\in A} g_\tau^a = 1$ بنابراین خواهیم داشت $\pi_{\boldsymbol{\tau}} = \sum_{a\in A} x_{\boldsymbol{\tau}}^a$

حال با جایگذاری $\{x_\tau^a\}$ به جای $\{\pi_\tau\}$ در $\mathcal{P}_1$ خواهیم داشت:

$$\mathcal{P}_2 : \min_{\boldsymbol{x},\eta} \bar{T} = \Big(\sum_{i=1}^{k} \frac{1}{\alpha_i} \sum_{\tau\in\mathcal{S}} \sum_{a\in A} \tau\{q_i\} \cdot x_\tau^a\Big) + T_p^0$$
$$\text{s.t.} \begin{cases} \nu_{loc}(\boldsymbol{x}) P_{loc} + \beta \nu_{tx}(\boldsymbol{x}) P_{tx} \le \bar{P}_{\max} \\ \Gamma(\boldsymbol{x}, \eta_i) =, \forall i \in \{1, \cdots, k\} \\ F_\tau(\boldsymbol{x}) = 0, \forall \tau = (i,m,n) \in \mathcal{S} \\ \sum_{\tau\in\mathcal{S}} \sum_{a\in A} = 1 \\ \eta_i \in [0,1], \forall i \in \{1, \cdots, k\} \\ x_\tau^a \ge 0, \forall \tau \in S, a \in A \end{cases} \tag{۱۴.۶-۴}$$

که در آن ν_{loc} و ν_{tx} به ترتیب احتمال فعالیت پردازنده و واحد ارسال را در یک واحد زمانی دلخواه مشخص می‌کنند و به ازای یک استراتژی داده شده قابل محاسبه اند. [۵] تابع $\Gamma(\boldsymbol{x}, \eta_i)$ بر اساس رابطه

[4] Occupation Measure

[۵] برای مشاهده روش محاسبه این دو پارامتر در قالب کد به پیوست ۲ مراجعه شود.

۸.۴-۴ می‌باشد و به صورت زیر محاسبه می‌شود:

$$\Gamma(x,\eta) = \eta \sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_2^i\cup\mathcal{S}_3^i\cup\mathcal{S}_4^i} x_\tau^a + 2\eta \sum_{\boldsymbol{\tau},\boldsymbol{a}\in S_5^i} x_\tau^a - \eta \sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_3^i\cup\mathcal{S}_5^i} x_\tau^a \qquad (۱۵.۶-۴)$$

و تابع $F_\tau(\boldsymbol{x})$ به صورت زیر تعریف می‌شود:

$$F_\tau(\boldsymbol{x}) = \sum_{\tau'\in\mathcal{S}}\sum_{a\in A} \tilde{\chi}_{\tau',\tau,a} x_{\tau'}^a - \sum_{a\in A} x_\tau^a \qquad (۱۶.۶-۴)$$

در رابطه فوق منظور از $\tilde{\chi}_{\tau',\tau,a}$ احتمال شرطی این است که به شرط اینکه در حالت τ' باشیم و کنش a انتخاب شده باشد، آنگاه به حالت τ' برویم و مطابق با رابطه ۱۷.۶-۴ بدست می آید. لازم به ذکر است که مقدار $\tilde{\chi}_{\tau',\tau,a}$ بر خلاف $\chi_{\tau,\tau'}$ نسبت به استراتژی تخلیه احتمالی g_τ^a مستقل است.

$$\tilde{\chi}_{\tau,\tau',\alpha} = P\left(\tau[t+1] = \tau' \mid \tau[t] = \tau \ \wedge v[t] = a\right) \qquad (۱۷.۶-۴)$$

در صورتی که مقادیر $\eta_0, \cdots, \eta_k$ معلوم باشد آنگاه مسئله $\mathcal{P}_2$ تبدیل به یک مسئله برنامه‌ریزی خطی می‌شود. با یافتن مقادیر جواب بهینه $\{x_\tau^a\}$ می‌توان استراتژی بهینه را طبق رابطه زیر بدست آورد:

$$g_\tau^{a*} = \frac{x_\tau^{a*}}{\sum_{a\in A} x_\tau^{a*}}, \forall\tau\in\mathcal{S}, a\in A \qquad (۱۸.۶-۴)$$

بنابراین جهت یافتن استراتژی بهینه برای یک سیستم تخلیه‌ی وظیفه کافی است که مسئله برنامه‌ریزی خطی حاصل از $\mathcal{P}_2$ را به ازای مقادیر مختلف $\eta_0, \cdots, \eta_k$ حل کرده تا استراتژی بهینه بدست بیاید. مراحل این فرآیند جستجو در الگوریتم ۱ به صورت خلاصه آمده است. در این الگوریتم تابع $splitRange$ تابعی است که با گرفتن یک بازه از اعداد حقیقی مانند R و پارامتر $precision$، تعداد $precision$ نمونه با فاصله‌های یکسان از بازه R را در قالب یک لیست بر می‌گرداند. منظور از $splitRange([0,1], precision)^k$ نیز ضرب دکارتی k نمونه از این لیست‌های خروجی در یک دیگر می‌باشد.

الگوریتم ۱.۴ الگوریتم جستجوی استراتژی تخلیه‌ی وظیفه بهینه

Require: $precision \geq 2$
1: $etaSettings \leftarrow splitRange([0,1], precision)^k$
2: $optimalPolicy = null$
3: **for each** $s \in etaSettings$ **do**
4: $(\eta_0, \cdots, \eta_k) \leftarrow s$
5: $solution \leftarrow solveLP(\eta_0, \cdots, \eta_k)$
6: **if** $optimalPolicy = null$ **or** $solution.delay < optimalPolicy.delay$ **then**
7: $optimalPolicy \leftarrow solution.policy$
8: **end if**
9: **end for**
10: **return** $optimalPolicy$

۴-۷ دو بهینه‌سازی برای الگوریتم جستجوی استراتژی

در این بخش دو بهینه‌سازی مختلف را به منظور بهبود عملکرد الگوریتم ۱.۴ معرفی می‌کنیم. این دو بهینه‌سازی در فریم‌ورک Kompute که در فصل پیش رو ارائه خواهد شد پیاده‌سازی شده‌اند.

۴-۷-۱ کاهش تعداد متغیرها

در مسئله بهینه‌سازی $\mathcal{P}_2$ تعداد $|S| \cdot |A|$ متغیر وجود دارد. این مقدار برای تعداد صف‌های کم (برای مثال $k \leq 3$) قابل اجرا می‌باشد اما با افزایش تعداد صف‌ها اجرای الگوریتم را بسیار زمان‌بر و یا غیرممکن می‌کند. یک بهینه‌سازی خیلی ساده ولی کارآمد که در [۱] به آن اشاره‌ای نشده است این است که می‌توان تمام متغیرهای مانند x_τ^a که کنش a جزو کنش‌های ممکن در τ نباشد را حذف کرد زیرا مقدار آنها در جواب مسئله همواره برابر صفر می‌باشد. برای مثال در جدول ۴-۳ مجموعه تمام کنش‌های سیستم تخلیه توصیف شده در جدول ۴-۱ به همراه امکان‌پذیری هر کنش در صورت حضور در حالت $\tau = ([3,0],0,1,0,1)$ مشخص شده است. همانطور که مشاهده می‌شود در این حالت فقط ۲ کنش از مجموعه ۹ کنش موجود در A امکان‌پذیر می‌باشند. کنش‌های ردیف ۲ و ۳ به دلیل مشغول بودن پردازنده امکان‌پذیر نمی‌باشند. کنش ردیف ۵ به دلیل خالی بودن صف وظایف نوع ۲ امکان‌پذیر نمی‌باشد. کنش‌های ردیف ۶ الی ۹ نیز به دلیل مشغول بودن پردازنده امکان پذیر نمی‌باشند. بنابراین می‌توانیم به سادگی ۷ متغیر متناظر این کنش‌ها را از مجموعه $\{x_\tau^a\}$ حذف کنیم

Row	Action	Is Possible
1	`NoOperation`	Yes
2	`AddToCPU(queueIndex = 1)`	No
3	`AddToCPU(queueIndex = 2)`	No
4	`AddToTransmissionUnit(queueIndex = 1)`	Yes
5	`AddToTransmissionUnit(queueIndex = 2)`	No
6	`AddToBothUnits(cpuQueueIndex = 1, tuQueueIndex = 1)`	No
7	`AddToBothUnits(cpuQueueIndex = 1, tuQueueIndex = 2)`	No
8	`AddToBothUnits(cpuQueueIndex = 2, tuQueueIndex = 1)`	No
9	`AddToBothUnits(cpuQueueIndex = 2, tuQueueIndex = 2)`	No

جدول ۴-۳: امکان پذیری کنش‌های مختلف در حالت $\tau = ([3, 0], 0, 1, 0, 1)$

بدون اینکه تغییری در جواب مسئله بهینه‌سازی $\mathcal{P}_2$ ایجاد شود.

۴-۷-۲ موازی‌سازی

الگوریتم ۱.۴ به گونه‌ای تعریف شده است که امکان موازی‌سازی و مقیاس‌پذیری[۶] آن به صورت خطی وجود دارد. به عبارت دیگر می‌توان مسئله برنامه‌ریزی خطی متناظر با هر مقداردهی از $\eta_0, \cdots, \eta_k$ را به یک هسته یا گره پردازشی خاص اختصاص داد. در شبیه‌سازی سناریو وظایف سبک و سنگین (رجوع شود به ۶-۲-۳) مشاهده شد که الگوریتم موازی‌سازی شده هنگام اجرا بر روی سروری با ۲۴ هسته و تقسیم‌بندی به ۲۴ رشته[۷] عملکردی معادل ۲۰ برابر سریع‌تر از حالت تک‌رشته[۸] دارد.

[6] Scaling

[7] Thread

[8] Single-thread

الگوریتم ۲.۴ الگوریتم جستجوی استراتژی تخلیه‌ی وظیفه بهینه موازی‌سازی شده

Require: $precision \geq 2$
Require: $threadCnt \geq 1$
1: **synchronized** $optimalPolicy = null$
2: **procedure** $findOptimalForEtaSettings(etaSettings)$
3: **for each** $s \in etaSettings$ **do**
4: $(\eta_0, \cdots, \eta_k) \leftarrow s$
5: $solution \leftarrow solveLP(\eta_0, \cdots, \eta_k)$
6: **if** $optimalPolicy = null$ **or** $solution.delay < optimalPolicy.delay$ **then**
7: $optimalPolicy \leftarrow solution.policy$
8: **end if**
9: **end for**
10: **end procedure**
11: $etaSettings \leftarrow splitRange([0, 1], precision)^k$
12: $etaBatches \leftarrow splitToBatches(etaSettings, threadCnt)$
13: **for each** $i \in 1 \cdots threadCnt$ **do**
14: $thread[i] = Thread\{findOptimalForEtaSettings(etaBatches[i])\}$
15: **end for**
16: **for each** $i \in 1 \cdots threadCnt$ **do**
17: $thread[i].start()$
18: **end for**
19: **for each** $i \in 1 \cdots threadCnt$ **do**
20: $thread[i].join()$
21: **end for**
22: **return** $optimalPolicy$

فصل ۵

پیاده‌سازی عملی

در این فصل چارچوب نرم‌افزاری با نام **کامپیوت**[۱] (Kompute) ارائه خواهیم کرد که با استفاده از آن می‌توان الگوریتم ارائه شده در فصل پیشین برای یافتن استراتژی تخلیه بهینه را به ازای پارامترهای محیطی مختلف اجرا کرد و عملکرد استراتژی تخلیه را با کمک شبیه‌سازی بررسی کرد. این چارچوب طبق یافته‌های ما اولین پیاده‌سازی متن‌باز در زمینه استراتژی تخلیه‌ی وظیفه ناهمگون در رایانش لبه‌ای است. در این فصل ابتدا توضیح مختصری در مورد نحوه کارکرد و معماری کامپیوت خواهیم داد و سپس برنامه‌های نمونه‌ای برای «پیدا کردن استراتژی تخلیه بهینه» و «شبیه‌سازی استراتژی تخلیه» ارائه خواهیم کرد. کامپیوت در زبان کاتلین[۲] نوشته شده است که زبان برنامه‌نویسی چندمنظوره‌ای است که نخستین بار توسط شرکت «جت برینز[۳]» ارائه شد. محبوب‌ترین نسخه این زبان نسخه ماشین مجازی جاوا می‌باشد. چند دلیل انتخاب کاتلین برای پیاده‌سازی پروژه‌ی فعلی عبارتند از:

- سادگی نحو زبان
- قابلیت‌های زیاد کتابخانه استاندارد
- پشتیبانی از واسط بومی جاوا[۴] به منظور حل سریع برنامه‌های خطی در زبان C++

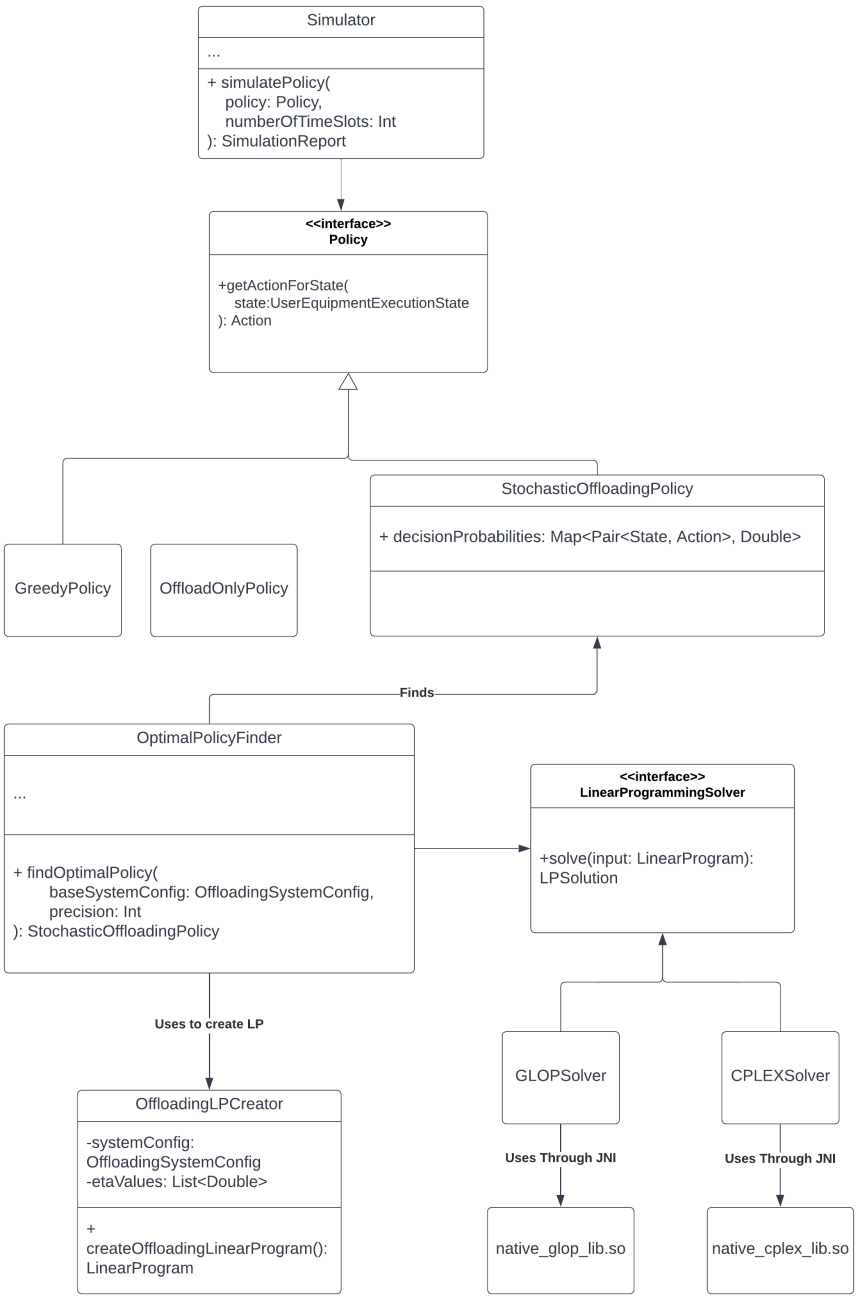
[1] https://github.com/dalisyron/Kompute

[2] Kotlin

[3] JetBrains

[4] Java Native Interface

معماری کلی کامپیوت در قالب یک کلاس دیاگرام در شکل ۵-۱ آورده شده است.



شکل ۵-۱: کلاس دیاگرام فریم‌ورک Kompute

۵-۱ مولفه‌های اصلی چارچوب Kompute

در این بخش به توضیح برخی از اجزای اصلی این چارچوب می‌پردازیم.

۵-۱-۱ واسط Policy

واسط Policy قردادی است که تمام استراتژی‌های تخلیه‌ی وظیفه باید آن را پیاده‌سازی کنند. همانطور که در فصل ۴ گفته شد، یک استراتژی تخلیه‌ی وظیفه دارد بسته به حالت فعلی سیستم، تصمیم بگیرد که چه کنشی برای اجرا در بازه زمانی فعلی انتخاب شود. این منطق در کامپیوت با استفاده از قطعه کد ۵-۱ تعریف شده است.

قطعه کد ۵-۱: واسط Policy

```
interface Policy {

   fun getActionForState(state: UserEquipmentExecutionState): Action
}
```

به عنوان نمونه برای پیاده‌سازی استراتژی تخلیه‌ی وظیفه «حریصانه-تخلیه اول[5]» کلاس وارث مطابق با قطعه کد ۵-۲ تعریف می‌شود. این استراتژی تخلیه در صورتی که بتواند هم تخلیه و هم پردازش محلی انجام دهد، هر دو را انجام خواهد داد و در صورتی که تنها یک وظیفه در صف‌های وظایف باشد آن وظیفه را تخلیه خواهد کرد.

قطعه کد ۵-۲: پیاده‌سازی استراتژی تخلیه‌ی وظیفه حریصانه-تخلیه اول

```
class GreedyOffloadFirstPolicy : Policy {

   override fun getActionForStateGreedy(state: UserEquipmentExecutionState): Action {
      if (state.averagePower() > state.pMax) return Action.NoOperation
      if (state.ueState.isCPUActive() && state.ueState.isTUActive()) {
         return Action.NoOperation
      }
      if (state.taskQueueLength.all { it == 0 }) return Action.NoOperation
      if (state.ueState.isCPUActive()) {
         return OffloadOnlyPolicy.getActionForState(state)
      }
      if (state.ueState.isTUActive()) {
         return LocalOnlyPolicy.getActionForState(state)
      }
      val nonEmptyIndices = state.taskQueueLength.indices.filter {
         state.taskQueueLength[it] > 0
      }
      require(nonEmptyIndices.isNotEmpty())
```

[5] Greedy-Offload First

```
        val queueIndices: Pair<Int, Int>? =
        state.ueState.getTwoRandomQueueIndicesForTwoTasks()
        if (queueIndices == null) {
            return Action.AddToTransmissionUnit(nonEmptyIndices[0])
        } else {
            return Action.AddToBothUnits(queueIndices.first, queueIndices.second)
        }
    }
}
```

۵-۱-۲ کلاس OffloadingLPCreator

این کلاس وظیفه ساخت مسئله برنامه‌ریزی خطی $\mathcal{P}_2$ که در رابطه ۴-۱۴.۶ بیان شد را دارد. بدین منظور این کلاس پنج شگرد زیر را تعریف می‌کند:

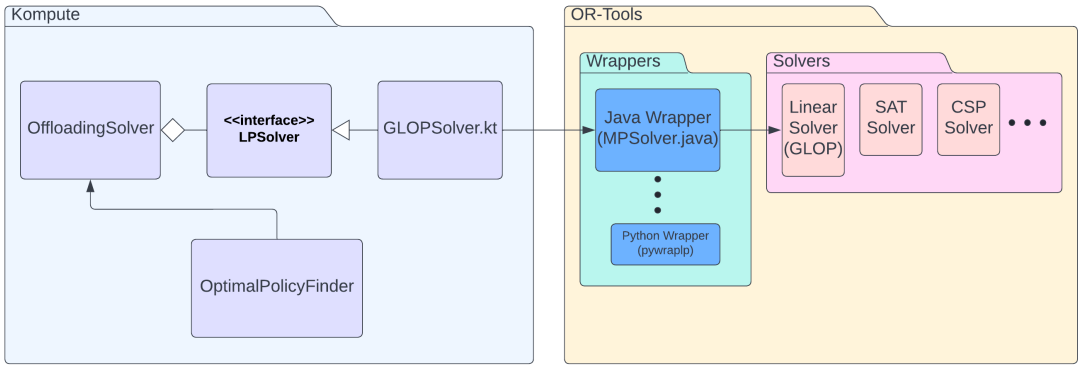
- `fun getObjectiveEquation(): EquationRow`
- `fun getEquation2(): EquationRow`
- `fun getEquations3(): List<EquationRow>`
- `fun getEquation4s(): List<EquationRow>`
- `fun getEquation5(): EquationRow`

که به ترتیب تابع هدف مسئله بهینه‌سازی و چهار شرط ذکر شده در ۴-۱۴.۶ را محاسبه می‌کنند. در نهایت با ترکیب این پنج تابع یک شی برنامه خطی توسط شگرد `createOffloadingLinearProgram` ایجاد می‌شود که به منظور محاسبه استراتژی تخلیه بهینه نیاز داریم که آن را حل کنیم. در ادامه به نحوه حل این برنامه خطی توسط کلاس‌های `Solver` می‌پردازیم.

۵-۱-۳ حل‌کننده خطی

مولفه‌های شرکت‌کننده در حل مسئله بهینه‌سازی استراتژی تخلیه در کامپیوت در شکل ۵-۲ مشخص شده اند. در پایین‌ترین لایه برای حل مسئله برنامه‌ریزی خطی از حل‌کننده خطی [۶] به نام GLOP

[6] Linear Solver



شکل ۵-۲: مولفه‌های شرکت‌کننده در حل مسئله خطی استراتژی تخلیه در Kompute

استفاده شده است. این حل‌کننده جزئی از پروژه متن‌باز OR-Tools است[۷] که توسط شرکت گوگل ارائه شده و نگه‌داری می‌شود. [۱۶] این حل‌کننده مانند اکثر حل‌کننده‌های سریع و مدرن، در زبان C++ نوشته شده است، اما احاطه‌گرهای آن در زبان‌های دیگر مانند پایتون، جاوا، و سی شارپ وجود دارد. در پروژه‌ی فعلی ، ما از احاطه‌گر زبان جاوا استفاده کرده‌ایم که در پروژه OR-Tools با استفاده از رابط بومی جاوا و در قالب کلاس `MPSolver` پیاده‌سازی شده است.

در Kompute کلاسی به نام GLOPSolver وجود دارد که با گرفتن یک شی از نوع برنامه خطی در دامنه Kompute، آن شی را به برنامه‌خطی قابل شناسایی برای کلاس MPSolver تبدیل می‌کند و در نهایت نتیجه حل برنامه‌خطی را بر می‌گرداند. کلاس MPSolver قابلیت پشتیبانی از برخی از حل‌کننده‌های دیگر به جز GLOP را نیز دارد. با این وجود، به دلیل متن‌باز بودن پروژه، از برخی از حل‌کننده‌های تجاری معروف مانند CPLEX[۸] پشتیبانی نمی‌کند. معماری Kompute به گونه‌ای طراحی شده است که امکان استفاده از هر حل‌کننده خطی در آن وجود داشته باشد. برای نمونه علاوه بر حل‌کننده پیش‌فرض GLOPSolver کلاسی با نام CPLEXSolver نیز در پروژه وجود دارد که در صورتی که نسخه تجاری CPLEX که دارای جواز معتبر باشد بر روی سیستم کاربر نصب باشد از آن حل‌کننده استفاده خواهد شد.

[7] https://github.com/google/or-tools

[8] https://www.ibm.com/analytics/cplex-optimizer

۵-۲ تعریف و حل یک مسئله تخلیه‌ی وظیفه نمونه در Kompute

در قطعه کد نمونه زیر، مسئله تخلیه‌ی وظیفه برای محیط رایانش لبه‌ای با دو صف[۹] حل شده است.

قطعه کد ۵-۳: تعریف و حل مسئله نمونه

```
fun main(args: Array<String>) {
   val systemConfig = OffloadingSystemConfig(
      userEquipmentConfig = UserEquipmentConfig(
         stateConfig = UserEquipmentStateConfig(
            taskQueueCapacity = 5,
            tuNumberOfPackets = listOf(1, 3),
            cpuNumberOfSections = listOf(7, 2),
            numberOfQueues = 2
         ),
         componentsConfig = UserEquipmentComponentsConfig(
            alpha = listOf,4.0) ,(9.0
            beta = ,90.0
            etaConfig = null,
            pTx = ,0.1
            pLocal = ,8.0
            pMax = 7.1
         )
      ),
      environmentParameters = EnvironmentParameters(
         nCloud = listOf(1, 1),
         tRx = ,5.0
      )
   )

   val optimalPolicy = RangedOptimalPolicyFinder.findOptimalPolicy(
      baseSystemConfig = systemConfig,
      precision = 10
   )
   /*
   // For multi-threaded execution use this instead:

   val optimalPolicy = ConcurrentRangedOptimalPolicyFinder(
      baseSystemConfig = systemConfig
   ).findOptimalPolicy(precision = 10, numberOfThreads = 8)
   */


   val decisionProbabilities: Map<StateAction, Double>
      = optimalPolicy.stochasticPolicyConfig.decisionProbabilities

   println(decisionProbabilities)
}
```

به این منظور ابتدا پارامترهای محیط تخلیه‌ی وظیفه در رایانش لبه‌ای را با استفاده از کلاس `OffloadingSystemConfig` مشخص می‌کنیم. سپس استراتژی بهینه را با استفاده از کلاس `RangedOptimalPolicyFinder` با دقت لازم پیدا می‌کنیم. در نهایت جواب خروجی به صورت توزیع احتمالی بر روی مجموعه $|S| \times |A|$ بدست می‌آید.

[۹] شرایط بر اساس تقسیم‌بندی وظایف به Heavy و Light در اینترنت اشیاء

۵-۳ شبیه‌سازی استراتژی‌های تخلیه‌ی وظیفه

در قطعه کد نمونه زیر استراتژی تخلیه بهینه به همراه سه استراتژی تخلیه پایه شبیه‌سازی شده‌اند و نتایج تاخیر و توان مصرفی گزارش شده است.

قطعه کد ۵-۴: شبیه‌سازی استراتژی‌های تخلیه‌ی وظیفه

```
fun main(args: Array<String>) {
   val baseSystemConfig: OffloadingSystemConfig = Mock.doubleQueueConfig()
   val alpha0Start = 01.0
   val alpha0End = 60.0
   val sampleCount = 30
   val simulationCycles = 1_000_000
   for (i in 0 until sampleCount) {
      val alpha0 = (alpha0Start + i * ((alpha0End - alpha0Start) / (sampleCount - 1)))
      val systemConfig = baseSystemConfig.withAlpha(0, alpha0)
      val optimalPolicy = RangedOptimalPolicyFinder.findOptimalPolicy(
         baseSystemConfig = systemConfig,
         precision = 10
      )
      val simulator = Simulator(systemConfig)
      val simulationResults = with(simulator) {
         listOf(
            simulatePolicy(LocalOnlyPolicy, simulationCycles),
            simulatePolicy(OffloadOnlyPolicy, simulationCycles),
            simulatePolicy(GreedyLocalFirstPolicy, simulationCycles),
            simulatePolicy(optimalPolicy, simulationCycles)
         )
      }
      val (localOnlyDelay,
         offloadOnlyDelay,
         greedyLocalFirstDelay,
         optimalDelay) = simulationResults.map { it.averageDelay }
      val (localOnlyAveragePower,
         offloadOnlyAveragePower,
         greedyLocalFirstAveragePower,
         optimalAveragePower) = simulationResults.map { it.averagePowerConsumption }
      println("$localOnlyDelay | " +
         "$offloadOnlyDelay | " +
         "$greedyLocalFirstDelay | " +
         "$optimalDelay")

      println("$localOnlyAveragePower | " +
         "$offloadOnlyAveragePower | " +
         "$greedyLocalFirstAveragePower | " +
         "$optimalAveragePower")
   }
}
```

در این مثال، به ازای مقادیر نرخ ورودی مختلف برای صف شماره یک (اندیس صفر)، با کمک کلاس `Simulator` معیارهای توان مصرفی و تاخیر محاسبه و گزارش شده است. پارامتر `simulationCycles` تعداد بازه‌های زمانی لازم برای شبیه‌سازی را مشخص می‌کند. پرواضح است که هر چه این مقدار بالاتر باشد، دقت شبیه‌سازی بالاتر خواهد بود.

فصل ۶

آزمایش و نتیجه

در این فصل قصد داریم عملکرد و درستی روش ارائه شده در فصول ۴ و ۵ را با کمک آزمون‌های مختلف و شبیه‌سازی بررسی کنیم. در بخش نخست این فصل، به بررسی صحت مدل تعریف شده در مسئله خواهیم پرداخت. بدین منظور تاخیر سرویس بدست آمده توسط مدل تعریف شده را با نتیجه اجرای شبیه‌سازی بر روی همان مدل مقایسه خواهیم کرد. به عبارتی به کمک شبیه‌سازی مدل را با «خودش» مقایسه خواهیم کرد. در بخش دوم، به مقایسه عملکرد استراتژی بدست آمده توسط روش پیشنهادی با سایر استراتژی‌ها می‌پردازیم. در این بخش چندین استراتژی رایج را به عنوان استراتژی‌های پایه تعریف می‌کنیم و عملکرد استراتژی تخلیه پیشنهادی را در برابر آنها با کمک شبیه‌سازی بررسی می‌کنیم.

۶-۱ بررسی صحت مدل

در این بخش بررسی خواهیم کرد که آیا مقدار تاخیر سرویس بدست آمده توسط شبیه‌سازی با مقدار تاخیر مشخص شده در جواب بهینه مسئله $\mathcal{P}_2$ (رابطه ۴-۱۴.۶) همخوانی دارد یا نه. بدین منظور نتایج الگوریتم جستجوی استراتژی تخلیه را در دو سناریو شبیه‌سازی مختلف بررسی خواهیم کرد.

۶-۱-۱ سناریو تک صف

در این آزمون محیطی با یک صف وظیفه در نظر گرفته شده است که ویژگی‌های آن در جدول ۶-۳ مشخص شده است. شبیه‌سازی به ازای ۲۹ مقدار مختلف η_0 صورت گرفته‌است که نتایج آن در مقایسه با مقدار محاسبه شده توسط مدل مسئله (برنامه خطی $\mathcal{P}_2$) در جدول ۶-۱ خلاصه شده است. تعداد سیکل‌های شبیه‌سازی در هر ۲۹ حالت برابر با 10^7 بوده است. همانطور که مشاهده می‌شود مقدار تاخیر سرویس محاسبه شده توسط مدل برنامه‌ریزی خطی، بسیار نزدیک به مقدار تاخیر بدست آمده توسط شبیه‌سازی می‌باشد.

۶-۱-۲ سناریو دو صف

در این آزمون محیطی با دو صف وظیفه در نظر گرفته شده است که ویژگی‌های آن در جدول ۶-۴ مشخص شده است. شبیه‌سازی به ازای ۲۲ مقداردهی مختلف به η_1, η_2 صورت گرفته‌است که نتایج آن در مقایسه با مقدار محاسبه شده توسط مدل مسئله در جدول ۶-۲ خلاصه شده است. تعداد سیکل‌های شبیه‌سازی در هر ۲۹ حالت برابر با 10^7 بوده است. همانطور که مشاهده می‌شود مقدار تاخیر سرویس محاسبه شده توسط مدل برنامه‌ریزی خطی، بسیار نزدیک به مقدار بدست آمده توسط شبیه‌سازی می‌باشد.

۶-۲ بررسی عملکرد در مقایسه با الگوریتم‌های پایه

در این بخش عملکرد استراتژی یافت شده توسط الگوریتم ۱.۴ را با چهار الگوریتم پایه زیر مقایسه می‌کنیم:

۱. استراتژی «فقط تخلیه»[1] که همه‌ی وظایف را تخلیه می‌کند

۲. استراتژی «حریصانه، تخلیه اول»[2] که در هر بازه زمانی اگر واحد ارسال یا پردازنده بیکار باشند به هر کدام از آنها یک وظیفه از صفی رندوم تخصیص می‌دهد و در صورتی که تنها یک وظیفه در صف باشد و مجبور به انتخاب بین تخلیه و اجرای محلی باشد، تخلیه را انتخاب می‌کند.

[1] Offload Only

[2] Greedy (Offload First)

η_1	Delay (model estimate)	Delay (simulation result)	Error
0.01	5.9403373	5.945382	0.0050447
0.02	5.9413582	5.9441002	0.002742
0.03	5.9873212	5.9979591	0.0106379
0.04	6.0332846	6.0445199	0.0112353
0.05	6.0792458	6.0772735	-0.0019723
0.06	6.1653313	6.1611952	-0.0041361
0.07	6.2608539	6.2771005	0.0162466
0.08	6.3563761	6.3485279	-0.0078482
0.09	6.4518981	6.4535551	0.001657
0.1	6.5474205	6.5470207	-0.0003998
0.11	6.6429429	6.6441015	0.0011586
0.12	6.7384654	6.7475386	0.0090732
0.13	6.8339881	6.8304343	-0.0035538
0.14	6.963416	6.967968	0.004552
0.15	7.117219	7.1221635	0.0049445
0.16	7.2710211	7.2660879	-0.0049332
0.17	7.4248235	7.4243839	-0.0004396
0.18	7.578626	7.5757626	-0.0028634
0.19	7.7324285	7.7334524	0.0010239
0.2	7.8862311	7.8823844	-0.0038467
0.21	8.0400334	8.0431362	0.0031028
0.22	8.193836	8.1896367	-0.0041993
0.23	8.3476384	8.3507161	0.0030777
0.24	8.5014409	8.50166	0.0002191
0.25	8.6793609	8.6778177	-0.0015432
0.26	8.9366602	8.9342996	-0.0023606
0.27	9.334054	9.3359713	0.0019173
0.28	9.9963099	9.9920628	-0.0042471
0.29	11.6247515	11.6247351	-0.0000164
		Variance	0.0000314
		Mean absolute difference	0.0041032

جدول ۶-۱: مقایسه میزان تاخیر بدست آمده از مدل و شبیه‌سازی در سناریو تک صف

η_1	η_2	Delay (model estimate)	Delay (simulation result)	Error
0	0	5.3055545	5.3043168	-0.0012377
0	0.2	4.5749717	4.5740879	-0.0008838
0	0.4	4.2116748	4.2124139	0.0007391
0	0.6	3.9532195	3.954735	0.0015155
0	0.8	3.6947642	3.6941737	-0.0005905
0	1	3.4702381	3.4711606	0.0009225
0.2	0	5.4240034	5.425082	0.0010786
0.2	0.2	4.9543158	4.9546973	0.0003815
0.2	0.4	4.7082897	4.7077916	-0.0004981
0.2	0.6	4.5023325	4.5033802	0.0010477
0.2	0.8	4.3225922	4.3218913	-0.0007009
0.2	1	4.3916784	4.3898362	-0.0018422
0.4	0	6.0300612	6.0294958	-0.0005654
0.4	0.2	5.6038029	5.6044667	0.0006638
0.4	0.4	5.3794469	5.3817302	0.0022833
0.4	0.6	5.1951134	5.1951965	0.0000831
0.4	0.8	5.1865812	5.1870742	0.000493
0.4	1	5.4041958	5.4026623	-0.0015335
0.6	0	7.4340974	7.4330417	-0.0010557
0.6	0.2	7.2298104	7.2298944	0.000084
0.6	0.4	7.5736237	7.5743988	0.0007751
0.6	0.6	8.7051662	8.7024916	-0.0026746
			Variance	0.0000314
			Mean absolute difference	0.0041032

جدول ۶-۲: مقایسه میزان تاخیر بدست آمده از مدل و شبیه‌سازی در سناریو دو صف

Parameter	M_1	L_1	β	P_{tx}	P_{loc}	P_{max}	C_1	t_{rx}
Value	1	17	0.4	1.0	0.8	1.6	1	0.0

جدول ۶-۳: پارامترهای محیط رایانش لبه‌ای در سناریو تک صف

۳. استراتژی «حریصانه، محلی اول»[۳] که در هر بازه زمانی اگر واحد ارسال یا پردازنده بیکار باشند به هر کدام از آنها یک وظیفه از صفی رندوم تخصیص می‌دهد و در صورتی که تنها یک وظیفه در صف باشد و مجبور به انتخاب بین تخلیه و اجرای محلی باشد، اجرای محلی را انتخاب می‌کند.

۴. استراتژی «فقط (اجرای) محلی»[۴]

۶-۲-۱ شبیه‌سازی تک صف

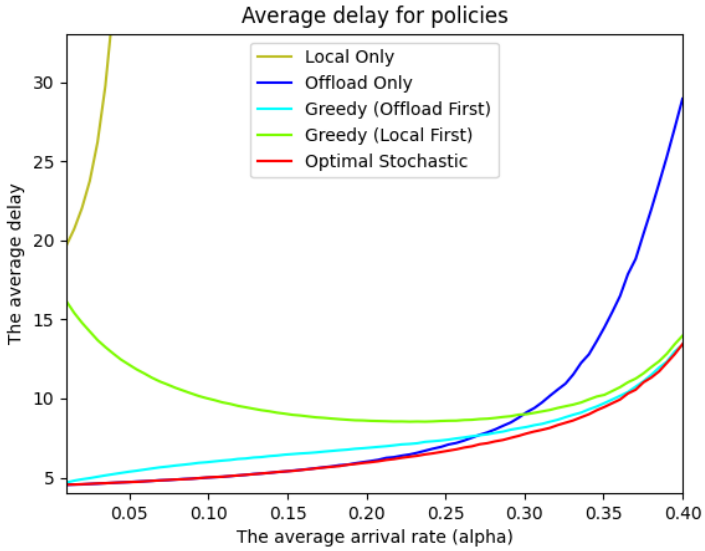
با توجه به اینکه روش ارائه شده توسط ما حالت گسترش یافته [۱] است، ابتدا محیط تست ارائه شده در آن پژوهش را برای تست الگوریتم در نظر می‌گیریم. پارامترهای این محیط در جدول ۶-۳ خلاصه شده اند. نتیجه این آزمایش در شکل ۶-۱ مشاهده می‌شود.
همانطور که مشاهده می‌شود استراتژی تخلیه تصادفی یافت شده از تمام الگوریتم‌های پایه بهتر عمل می‌کند و شکل منحنی‌های نمودار با [۱] مطابقت دارد.

۶-۲-۲ شبیه‌سازی دو صف با یک صف ثابت در سناریو سبک و سنگین

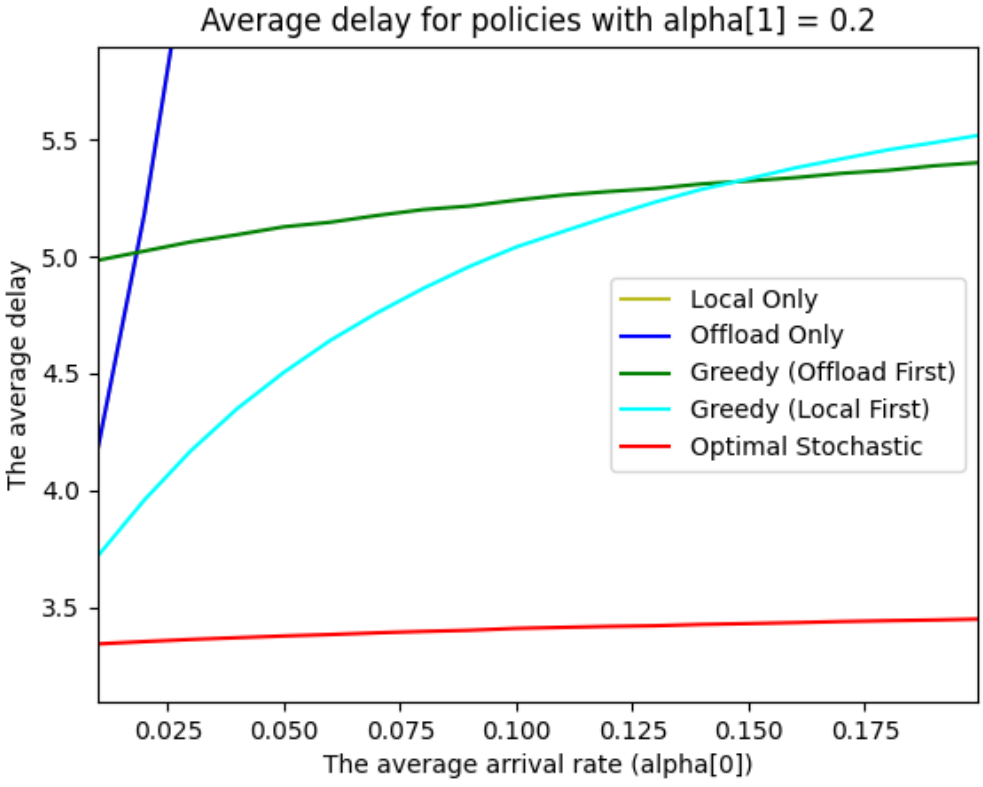
در این قسمت سناریوی تست به این گونه است که میزان تاخیر به ازای مقادیر مختلف نرخ ورود برای صف شماره یک و مقدار ثابت نرخ ورود برای صف شماره دو مشاهده می‌شود. پارامترهای محیطی در نظر گرفته شده در جدول ۶-۴ به طور خلاصه آمده است. همانطور که مشاهده می‌شود استراتژی تخلیه بهینه بسیار بهتر از الگوریتم‌های پایه عمل می‌کند. دلیل اصلی این تفاوت زیاد (نسبت به تفاوت کم در سناریو با یک صف در بخش قبل) عدم هوشمندی استراتژی‌های حریصانه در انتخاب نوع وظیفه تخصیص داده شده به پردازنده و واحد ارسال است. به عبارت دیگر انتخاب تصادفی نوع

[3] Greedy (Local First)

[4] Local Only



شکل ۶-۱: تاخیر سرویس به ازای نرخ ورود در حالت تک صف



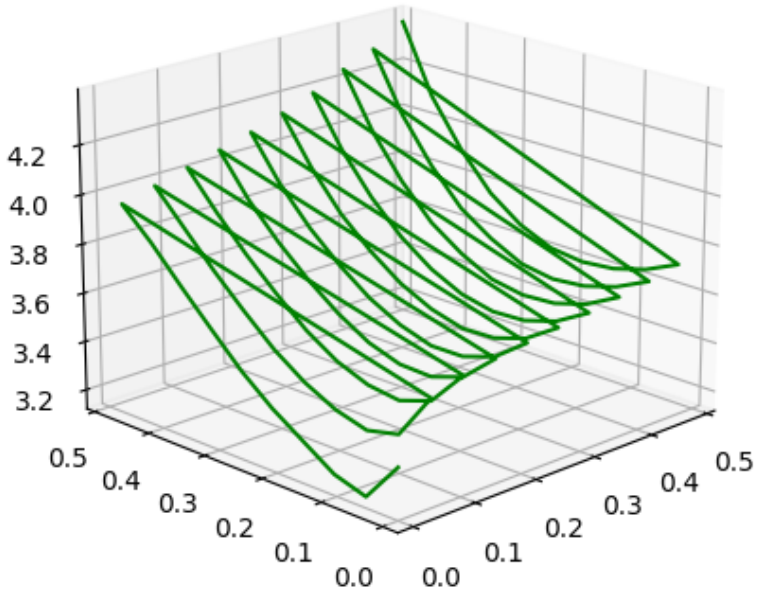
Parameter	M_1	M_2	L_1	L_2	C_1	C_2	β	P_{tx}	P_{loc}	P_{max}	t_{rx}
Value	1	3	7	2	1	1	0.95	1	0.8	1.6	0

جدول ۶-۴: پارامترهای محیط رایانش لبه‌ای در سناریو دو صف با یک صف ثابت

وظیفه فرستاده شده به پردازنده و واحد ارسال در الگوریتم‌های حریصانه باعث می‌شود که در شرایطی که تفاوت زیادی بین نوع وظایف وجود دارد (مانند سناریو سبک و سنگین) این الگوریتم‌ها عملکرد خیلی بدی داشته باشند. این در حالی است که در حالت تک صف انتخاب بین انواع وظیفه مطرح نبوده است و تنها عامل برای عملکرد غیربهینه‌ی استراتژی‌های حریصانه، عدم زمانبندی درست وظایف بوده است.

۶-۲-۳ شبیه‌سازی دو صف متغیر وظایف سبک و سنگین

در این قسمت مقدار تاخیر سرویس به ازای مقادیر مختلف نرخ ورود به هر دو صف محاسبه شده است. پارامترهای سیستمی این سناریو در جدول ۶-۵ آمده است. همانطور که مشاهده می‌شود استراتژی بهینه در بازه $\alpha_1, \alpha_2 \in [0, 0.4]$ عملکرد قابل قبول دارد.



شکل ۶-۲: تاخیر میانگین بر حسب نرخ ورود α_1 و α_2

Parameter	M_1	M_2	L_1	L_2	C_1	C_2	β	P_{tx}	P_{loc}	P_{max}	t_{rx}
Value	1	3	7	2	1	1	0.95	1	0.8	1.6	0

جدول ۶-۵: پارامترهای محیط رایانش لبه‌ای در سناریو دو صف متغیر

۶-۲-۴ شبیه‌سازی سه صف وظیفه

در این قسمت عملکرد الگوریتم ارائه شده در شرایطی که سه صف وجود دارد بررسی شده است. پارامترهای محیط رایانش لبه‌ای در جدول ۶-۷ آورده شده است. با توجه به اینکه رسم نمودار در شرایط چهار بعدی امکان پذیر نیست از مفهومی به نام آزمون «کارآمدی» استفاده می‌کنیم. مفهوم کارآمدی را اینگونه تعریف می‌کنیم که یک استراتژی کارآمد است اگر احتمال پر بودن یک یا چند صف در سیستم از $\frac{1}{|S|}$ کمتر باشد. در این آزمایش، کارآمدی استراتژی‌های مختلف را به ازای ۱۰۰۰ نمونه مختلف در بازه‌های $\alpha_1, \alpha_2, \alpha_3 \in [0, 0.2]$ تست کردیم که نتایج آن در جدول ۶-۶ مشاهده می‌شود.

Policy	Optimal	Local Only	Greedy (Local First)	Greedy (Offload First)	Offload Only
Effectiveness	100.0%	8.5%	80.3%	79.3%	21.6%

جدول ۶-۶: درصد کارآمدی استراتژی‌ها

Parameter	M_1	M_2	M_3	L_1	L_2	L_3	C_1	C_2	C_3	β	P_{tx}	P_{loc}	P_{max}	t_{rx}
Value	1	3	2	4	2	3	1	1	2	0.95	1	0.8	1.6	0.5

جدول ۶-۷: پارامترهای محیط رایانش لبه‌ای در سناریو سه صف

۶-۳ تست کارآیی

یک نکته که در دو بخش پیشین به آن اشاره‌ای نشد کارایی الگوریتم ارائه شده از نظر زمان اجرا و حافظه مصرفی می‌باشد. در آزمایش‌های بخش پیشین تعداد صف‌ها ۳ یا کمتر در نظر گرفته شده بود که اجرای الگوریتم مسئله را به راحتی میسر می‌ساخت. دلیل این امر این است که با افزایش تعداد صف‌ها، فضای حالت مسئله به صورت نمایی بزرگ خواهد شد. در جدول ۶-۸ تعداد حالت‌های زنجیره مارکوف $|S|$ به همراه زمان اجرا و حافظه مصرفی لازم جهت حل مسئله آورده شده است. برای تفسیر راحت‌تر نتایج آزمایش، تعداد بسته‌ها و قسمت‌های تمام صف‌ها برابر مقدار ثابت $L_i = M_i = 2$ در نظر گرفته شده است. البته در شرایط واقعی قطعا تعداد قسمت‌ها و بسته‌های هر صف متفاوت خواهند بود. زیرا در غیر این صورت صف‌های با ویژگی‌های یکسان را می‌توان به یک صف با نرخ ورود مجموع تبدیل کرد. پردازنده استفاده شده در این آزمایش Intel® Xeon® Processor E3-1220 بوده است. حل‌کننده خطی استفاده شده GLOP بوده است[۵]. طول هر وظیفه برابر با $Q = 6$ در نظر گرفته شده است. همانطور که مشاهده می‌شود زمان اجرای الگوریتم به صورت نمایی افزایش می‌یابد و مسئله فقط برای تعداد صف‌های کمتر از ۵ در زمان قابل قبول حل می‌شود. این تعداد در محیط‌های با تنوع وظایف نسبتا کم مانند سناریو «سبک» «سنگین» که پیشتر بیان شد، انتزاع قابل قبولی از فضای مسئله ارائه می‌دهد و عملکرد بهتری از حالت تک وظیفه دارد. در فصل پیش رو چندین ایده که می‌تواند در کاهش فضای حالت مسئله و بهبود عملکرد الگوریتم موثر باشد، جهت پژوهش بیشتر ارائه شده اند.

Number of queues	State count ($\|S\|$)	Running time
1	14	80ms
2	147	433ms
3	1372	7003 ms
4	100842	24164 seconds ($\sim$7 Hours)

جدول ۶-۸: زمان اجرا و اندازه فضای حالت به ازای تعداد صف $k = 1, 2, 3, 4$

[۵] در آزمایش‌های انجام شده مشاهده شد که GLOP عملکرد بهتری نسبت به CPLEX دارد و به این دلیل انتخاب گردید

فصل ۷

جمع‌بندی و پیشنهادها

در پروژه‌ی فعلی روشی برای بدست آوردن استراتژی تخلیه‌ی وظیفه با تاخیر کمینه در شرایط حضور چندین نوع وظیفه در محیط رایانش لبه‌ای معرفی شد. همچنین چارچوب نرم‌افزاری جدیدی معرفی شد که قادر به محاسبه استراتژی تخلیه‌ی وظیفه بهینه و شبیه‌سازی آن می‌باشد. روش ارائه شده در فصل شبیه‌سازی به طور جامع تحت آزمایش قرار گرفت و عملکرد آن بررسی شد. با استفاده از آزمایش‌های شبیه‌سازی ابتدا ثابت شد که مدل در نظر گرفته شده مسئله تعریف شده را به درستی حل می‌کند و نتایج شبیه‌سازی با مقادیر بدست آمده توسط مدل همخوانی دارد. پس از آن با استفاده از شبیه‌سازی، عملکرد روش ارائه شده را با سایر روش‌های تخلیه‌ی وظیفه مقایسه کردیم. روش ارائه شده این پتانسیل را دارد که نحوه زمان‌بندی وظایف در محیط‌های با وظایف گوناگون مانند اینترنت اشیاء را به طور قابل توجهی بهبود ببخشد. با این حال چندین محدودیت در پروژه‌ی فعلی وجود دارد، که رفع آنها نیاز به پژوهش بیشتر دارد.

۷–۱ بهبود کارآیی الگوریتم

یک محدودیت اصلی در روش ارائه شده، افت کارآیی الگوریتم با افزایش تعداد صف‌ها به دلیل انفجار فضای حالت می‌باشد. به عبارت دیگر با افزایش تعداد متغیرهای مسئله برنامه‌ریزی خطی $\mathcal{P}_2$ (رابطه ۴-۱۴.۶) زمان اجرای الگوریتم به صورت تصاعدی بالا می‌رود. در بخش ۴-۷-۱ روشی ارائه شد که

تا حدی فضای حالت مسئله را کاهش می‌داد، با این حال همانطور که در نتایج شبیه‌سازی مشاهده شد، الگوریتم ارائه شده همچنان برای طول صف‌های بیشتر از ۵ کارآیی ندارد. در این بخش دو ایده مختلف را ارائه می‌کنیم که با پژوهش دقیق درباره آنها شاید بتوان عملکرد الگوریتم را بهبود داد.

۷-۱-۱ حذف تک‌کنش‌ها

یک ایده ممکن برای التیام مشکل انفجار فضای حالت، حذف «تک کنش» ها از فضای مسئله ($|S| \times |A|$) می‌باشد. به طور دقیق‌تر کنش a را برای حالت τ یک **تک‌کنش** می‌نامیم اگر تنها کنش ممکن در حالت τ باشد. با توجه به اینکه کنش `NoOperation` در همه حالت‌ها وجود دارد، تک کنش‌ها همواره متناظر با `NoOperation` می‌باشند.

پیشتر در بخش ۴-۷-۱ به این موضوع اشاره شد که می‌توان متغیرهایی که متناظر با کنش‌های غیر ممکن هستند را از مسئله بهینه‌سازی $\mathcal{P}_2$ حذف کرد. با استدلالی مشابه این احتمال وجود دارد که بتوان متغیرهایی که متناظر با تنها کنش ممکن در یک حالت هستند را از الگوریتم برنامه‌ریزی خطی حذف کرد، زیرا احتمال انتخاب کنش‌های متناظر با چنین متغیرهایی همواره ۱ (قطعی) می‌باشد و مقدار آن متغیرها در تعیین استراتژی بهینه تاثیری ندارد. با دقت در فضای حالت مسئله می‌توان دریافت که بسیاری از حالت‌های فضای مسئله دارای تک‌کنش می‌باشند. برای مثال هر حالتی که پردازنده و واحد ارسال هر دو در آن مشغول باشند از این نوع خواهد بود. فرآوانی چنین حالت‌هایی در فضای حالت مسئله بدین معنی است که حذف آنها می‌تواند کارآیی الگوریتم ارائه شده را به طور قابل توجهی بهبود بخشد. برای مثال در جدول . درصد حالت‌های محیط تخلیه بیان شده در جدول ۶-۴ که دارای تک‌کنش هستند مشخص شده است. با این حال حذف تک‌کنش‌ها از لیست متغیرهای مسئله، بر خلاف بهینه‌سازی ۴-۷-۱ ساختار زنجیره مارکوف را دگرگون خواهد، بنابراین احتمالا نیازمند تغییر توابع انتقال و تغییر شروط رابطه ۴-۱۴.۶ خواهد بود. پژوهش بیشتر در این زمینه قطعا مشکل انفجار فضای حالت را حل نخواهد، اما امید می‌رود که تعداد صف‌های قابل پشتیبانی در روش ارائه شده را به طور قابل توجهی افزایش دهد.

۷-۱-۲ اعمال جریمه برای اتلاف وظیفه

یک راه بنیادی‌تر برای رفع مشکل انفجار فضای حالت این است که کلا مدل حالت (τ) مسئله را اینگونه تغییر دهیم که هر صف ظرفیتی برابر با ۲ داشته باشد. طبیعتا اعمال چنین محدودیتی در مسئله فعلی ممکن نیست زیرا فرض کرده‌ایم که اتلاف صورت نمی‌گیرد، به این صورت که حضور در حالت‌های با صف پر ($q_i = Q$) را به منزله غیر کارامد بودن استراتژی در نظر گرفتیم (رجوع شود به بخش ۶-۲-۴). اما می‌توان تغییری در مدل اعمال کرد که هزینه اتلاف وظیفه را نیز در نظر بگیرد. به طور دقیق‌تر مدل برنامه‌ریزی خطی می‌تواند به ازای یک استراتژی تخلیه داده شده و نرخ ورود $\alpha_1 \cdots \alpha_k$ احتمال وقوع اتلاف وظیفه در صف با طول ۲ را در نظر بگیرد. در چنین حالتی می‌توان به میزان مشخصی مانند Ω جریمه تاخیر در تابع هدف در نظر گرفت. بدین صورت حل کننده خطی در راستای کم کردن تاخیر، سعی خواهد کرد که احتمال وقوع اتلاف وظیفه را در سامانه پایین بیاورد. با استفاده از این تغییر فضای حالت مسئله به طور قابل توجهی کوچک خواهد شد به طوری که اگر در روش قدیمی $|S_1|$ حالت وجود داشته باشد، در روش جدید $\frac{|S_1| \cdot 2^k}{Q^k}$ حالت وجود خواهد داشت.

۷-۲ قراردهی استراتژی تخلیه

یک موضوع مهم که در پروژه‌ی فعلی به آن پرداخته نشد، نحوه قراردهی استراتژی تخلیه یا به عبارتی «سازوکار تنظیم استراتژی در دستگاه کاربر» است. همانطور که در شبیه‌سازی های انجام شده در فصل ۶ مشاهده شد، الگوریتم محاسبه استراتژی تخلیه بهینه به منابع محاسباتی زیادی نیاز دارد. طبیعتا انجام چنین محاسباتی بر روی دستگاه کاربر که موجودیتی مانند تلفن همراه یا اینترنت اشیاء است عملی نخواهد بود. چه بسا که در پروژه‌ی فعلی نیز برای تمام شبیه‌سازی‌ها از سروری قدرتمند با ۲۴ هسته پردازشی استفاده شد. در سطح بالا، یک راه حل برای قراردهی روش پیشنهادی در پروژه، استفاده از معماری ای مشابه با شبکه‌های مبتنی بر نرم افزار[1] می‌باشد. برای مثال می‌توان در هر سرور رایانش لبه‌ای فرآیندی را قراردهی کرد، که هر n ساعت یک بار، اطلاعات محیطی (مانند نرخ ورود وظایف هر نوع) را از دستگاه‌های کاربر سرویس‌گیرنده بگیرد و متناسب با شرایط هر محیط، استراتژی تخلیه بهینه را محاسبه کرده و برای دستگاه کاربر مربوطه ارسال کند. با این وجود، پیاده‌سازی کارآمد

[1] Software Defined Networks

چنین سازوکاری، نیازمند پژوهش بیشتر می‌باشد.

پیوست ۱ – توابع انتقال حالت

تابع انتقال حالت به ازای کنش ورودی

```
fun getNextStateRunningAction(
   sourceState: UserEquipmentState,
   action: Action
): UserEquipmentState {
   return when (action) {
      is Action.NoOperation → {
         sourceState
      }
      is Action.AddToCPU → {
         getNextStateAddingToCPU(sourceState, action.queueIndex)
      }
      is Action.AddToTransmissionUnit → {
         getNextStateAddingToTU(sourceState, action.queueIndex)
      }
      is Action.AddToBothUnits → {
         getNextStateAddingToBothUnits(
            sourceState,
            action.cpuTaskQueueIndex,
            action.transmissionUnitTaskQueueIndex
         )
      }
   }
}
```

تابع انتقال حالت پایه

```
fun getNextStateAddingToCPU(
   sourceState: UserEquipmentState,
   queueIndex: Int
): UserEquipmentState {
   require(sourceState.cpuState == 0)
   require(sourceState.taskQueueLengths[queueIndex] > 0)

   val updatedLengths = sourceState.taskQueueLengths.decrementedAt(queueIndex)

   return sourceState.copy(
      taskQueueLengths = updatedLengths,
      cpuState = -1,
      cpuTaskTypeQueueIndex = queueIndex
   )
}
```

تابع انتقال حالت با کنش ارسال توسط واحد ارسال

```
fun getNextStateAddingToTU(
   sourceState: UserEquipmentState,
   queueIndex: Int
): UserEquipmentState {
   require(sourceState.tuState == 0)
   require(sourceState.taskQueueLengths[queueIndex] > 0)

   val updatedLengths = sourceState.taskQueueLengths.decrementedAt(queueIndex)

   return sourceState.copy(
      taskQueueLengths = updateLengths,
      tuState = 1,
      tuTaskTypeQueueIndex = queueIndex
   )
}
```

تابع انتقال حالت با کنش اجرا و ارسال به طور همزمان

```
fun getNextStateAddingToBothUnits(
   sourceState: UserEquipmentState,
   cpuQueueIndex: Int,
   tuTaskQueueIndex: Int
): UserEquipmentState {
   if (cpuQueueIndex == tuTaskQueueIndex) {
      require(sourceState.taskQueueLengths[cpuQueueIndex] > 1)
   } else {
      require(sourceState.taskQueueLengths[cpuQueueIndex] > 0)
      require(sourceState.taskQueueLengths[tuTaskQueueIndex] > 0)
   }
   return getNextStateAddingToCPU(
      getNextStateAddingToTU(sourceState, tuTaskQueueIndex),
      cpuQueueIndex
   )
}
```

پیوست ۲ – تابع ساخت شرط حداکثر توان مصرفی در برنامه خطی

تابع ساخت شرط حداکثر توان مصرفی

```
fun getEquation2(): EquationRow {
   val pLoc = systemConfig.pLoc
   val pTx = systemConfig.pTx
   val beta = systemConfig.beta
   val rhsEquation2 = systemConfig.pMax
   val coefficients = mutableListOfZeros(indexMapping.variableCount)

   indexMapping.coefficientIndexByStateAction.forEach { (stateAction, index) →
      val (state, action) = stateAction
      var coefficientValue = 0.0

      if (state.isTUActive()
      || (action is Action.AddToTransmissionUnit
      || action is Action.AddToBothUnits)) {
         coefficientValue += beta * pTx
      }

      if (state.isCPUActive()
      || (action is Action.AddToCPU)
      || (action is Action.AddToBothUnits)) {
         coefficientValue += pLoc
      }

      coefficients[index] = coefficientValue

   }

   return EquationRow(
      coefficients = coefficients,
      rhs = rhsEquation2,
      type = EquationRow.Type.LessThan
   )
}
```

پیوست ۳ – نحوه محاسبه ماتریس انتقال

در این بخش نحوه محاسبه درایه‌های ماتریس انتقال $\chi_{\tau,\tau'}$ به ازای حالت ورودی τ در قالب کد شرح داده شده است. همانطور که در بخش ۴-۲ گفته شد، درایه‌های ماتریس انتقال معادل «یال» های گراف زنجیره می‌باشند. بنابراین هدف ما پیدا کردن یال‌های گراف با مبدا τ به همراه وزن آنها می‌باشد. به این منظور ابتدا با کمک تابع زیر کنش‌های ممکن را برای حالت ورودی پیدا می‌کنیم:

تابع محاسبه کنش‌های ممکن به ازای حالت داده شده

```
override fun getPossibleActions(state: UserEquipmentState): List<Action> {
   val result = mutableListOf<Action>(Action.NoOperation)
   if (state.isCPUActive() && state.isTUActive()) return result
   val nonEmptyQueueIndices = state.taskQueueLengths.indices.filter {
      state.taskQueueLengths[it] > 0
   }
   if (!state.isCPUActive())
      for (queueIndex in nonEmptyQueueIndices) {
         if (config.limitation[queueIndex] ≠ StateManagerConfig.Limitation.OffloadOnly) {
            result.add(Action.AddToCPU(queueIndex))
         }
      }
   if (!state.isTUActive()) {
      for (queueIndex in nonEmptyQueueIndices) {
         if (config.limitation[queueIndex] ≠ StateManagerConfig.Limitation.LocalOnly) {
            result.add(Action.AddToTransmissionUnit(queueIndex))
         }
      }
   }
   if (!state.isTUActive() && !state.isCPUActive()) {
      for (i in nonEmptyQueueIndices) {
         for (j in nonEmptyQueueIndices) {
            if (i == j && state.taskQueueLengths[i] < 2) continue
            if (config.limitation[i] ≠ StateManagerConfig.Limitation.OffloadOnly
            && config.limitation[j] ≠ StateManagerConfig.Limitation.LocalOnly) {
               result.add(Action.AddToBothUnits(i,j))
            }
         }
      }
   }
   return result.sorted()
}
```

در مرحله بعد می‌بایست به ازای هر جفت حالت و کنش (τ, a)، مجموعه حالات ممکن در صورت حضور در حالت τ و انتخاب کنش a را محاسبه کنیم. به این منظور از تابع زیر استفاده می‌کنیم:

تابع محاسبه کنش‌های ممکن به ازای حالت داده شده

```
fun getTransitionsForAction(state: UserEquipmentState, action: Action): List<Transition> {
   checkStateAgainstLimitations(state)
   val stateAfterAction = getNextStateRunningAction(state, action).let {
      if (it.isCPUActive()) getNextStateAdvancingCPU(it) else it
   }
   checkStateAgainstLimitations(stateAfterAction)
   val transitions: MutableList<Transition> = mutableListOf()
   val notFullIndicesAfterAction = (stateAfterAction.taskQueueLengths.indices).filter {
      val queueLengths = stateAfterAction.taskQueueLengths[it]
      queueLengths < config.userEquipmentStateConfig.taskQueueCapacity
   }
   if (notFullIndicesAfterAction.isEmpty()) {
      if (stateAfterAction.isTUActive()) {
         transitions.add(Transition(
            source = state,
            dest = getNextStateAdvancingTU(stateAfterAction),
            transitionSymbols = listOf(listOf(action, ParameterSymbol.Beta))))
         transitions.add(Transition(state, stateAfterAction,
            listOf(listOf(action, ParameterSymbol.BetaC))))
      } else {
         transitions.add(Transition(state, stateAfterAction, listOf(listOf(action))))
      }
   } else {
      val taskArrivalMappings = getAllSubsets(notFullIndicesAfterAction.size)
      for (mapping in taskArrivalMappings) {
         val addTaskSymbols = mapping.mapIndexed { index, taskArrives →
            if (taskArrives)
               ParameterSymbol.Alpha(notFullIndicesAfterAction[index])
            else
               ParameterSymbol.AlphaC(notFullIndicesAfterAction[index])
         }
         val destState = getNextStateAddingTasksBasedOnMapping(
            stateAfterAction, mapping, notFullIndicesAfterAction
         )
         if (stateAfterAction.isTUActive()) {
            transitions.add(Transition(
               source = state,
               dest = getNextStateAdvancingTU(destState),
               transitionSymbols = listOf(
                  listOf(action, ParameterSymbol.Beta) + addTaskSymbols))
               )
            transitions.add(Transition(
               source = state,
               dest = destState,
               transitionSymbols = listOf(
                  listOf(action, ParameterSymbol.BetaC) + addTaskSymbols))
               )
         } else {
            transitions.add(Transition(state,destState, listOf(
               listOf(action) + addTaskSymbols)))
         }
      }
   }
   return transitions
}
```

در نهایت با ترکیب دو تابعی که تعریف شد می‌توانیم تابع سومی بنویسیم که تمام یال‌های با مبدا τ را پیدا کند:

تابع محاسبه یال‌های زنجیره به ازای حالت مبدا ورودی

```
fun getEdgesForState(state: UserEquipmentState): List<Edge> {
   return getPossibleActions(state)
   .map { action →
      getTransitionsForAction(state, action)
   }
   .flatten().map { it.toEdge() }
}
```

مراجع

[1] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in 2016 IEEE International Symposium on Information Theory (ISIT), pp.1451–1455, 2016.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol.3, no.5, pp.637–646, 2016.

[3] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing iot service delay via fog offloading," IEEE Internet of Things Journal, vol.5, no.2, pp.998–1010, 2018.

[4] H. Tran-Dang and D.-S. Kim, "Frato: Fog resource based adaptive task offloading for delay-minimizing iot service provisioning," IEEE Transactions on Parallel and Distributed Systems, vol.32, no.10, pp.2491–2508, 2021.

[5] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," ACM Comput. Surv., vol.52, feb 2019.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys '10, (New York, NY, USA), p.49□62, Association for Computing Machinery, 2010.

[7] G. Hu, Y. Jia, and Z. Chen, "Multi-user computation offloading with d2d for mobile edge computing," in 2018 IEEE Global Communications Conference (GLOBECOM), pp.1–6, 2018.

[8] X. Meng, W. Wang, and Z. Zhang, "Delay-constrained hybrid computation offloading with cloud and fog computing," IEEE Access, vol.5, pp.21355–21367, 2017.

[9] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," IEEE Transactions on Vehicular Technology, vol.67, no.1, pp.44–55, 2018.

[10] A. Shakarami, M. Ghobaei-Arani, M. Masdari, and M. Hosseinzadeh, "A survey on the computation offloading approaches in mobile edge/cloud computing environment: A stochastic-based perspective," Journal of Grid Computing, vol.18, pp.639–671, Dec 2020.

[11] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," IEEE Internet of Things Journal, vol.6, no.2, pp.3864–3872, 2019.

[12] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," IEEE Journal on Selected Areas in Communications, vol.33, no.12, pp.2510–2523, 2015.

[13] Z. Jiang and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," IEEE Access, vol.3, pp.2306–2316, 2015.

[14] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," IEEE Transactions on Wireless Communications, vol.12, no.9, pp.4569–4581, 2013.

[15] A.-E. M. Taha, N. A. Ali, and H. S. Hassanein, Frame-Structure and Node Identification, pp.147–160. 2011.

[16] 2022.

[17] X. Chen, T. Chen, Z. Zhao, H. Zhang, M. Bennis, and Y. JI, "Resource awareness in unmanned aerial vehicle-assisted mobile-edge computing systems," in 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), pp.1–6, 2020.

Abstract:

Edge computing is a distributed computing paradigm that seeks to provide users with lower response times, lower power consumption, and mobility management by bringing computing resources closer to the network edge. Since its introduction, edge computing and its standard implementations, such as Multi-access Edge Computing, have faced one important challenge: How to design efficient task offloading policies?

Furthermore, with the rapid growth of the smartphone and IoT industry, many new types of applications have been introduced to the internet, each having different resource needs. Thus, taking into account the heterogeneity of user tasks becomes an essential factor when designing task offloading policies for edge computing environments.

This paper introduces a method for finding the delay-optimal task offloading policy under the power consumption constraint. The method consists of two steps. First, the offloading system is modeled using Discrete-time Markov Chains. Then, an algorithm based on linear programming is used to find the optimal task offloading policy for the created model. In addition to discussing the problem mathematically, we introduce a new software framework, written in the Kotlin language, which allows users to find the optimal task offloading policy for a given system. This framework can also benchmark the optimal policy's effectiveness using simulation. The current paper is based on [1] and uses a similar method to that research.

Keywords: Task Offloading, Edge Computing, Markov Chains, Linear Programming, Cloud Computing

Iran University of Science and Technology

Computer Engineering Department

A multi-user computation offloading policy to minimize average latency for IoT devices

Bachelor of Computer Engineering Final Project

By:

Mohammadmobin Dariushhamedani

Supervisor:

Dr. Reza Entezari-Maleki

June 2022