دانشکده مهندسی کامپیوتر

استراتژی تخلیه محاسبات تصادفی برای وظایف ناهمگون

پروژه 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 : استراتژی تخلیه محاسبات تصادفی برای وظایف ناهمگون

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 یک الگوی محاسبات توزیع شده است که با نزدیک کردن منابع پردازشی به لبه شبکه، سعی دارد تا مزایایی مانند زمان پاسخگویی کمتر، مصرف باتری کمتر و تحرک‌پذیری را برای کاربران به ارمغان بیاورد. از زمان معرفی رایانش لبه‌ای و استانداردهای معروف آن مانند رایانش لبه‌ای چند دسترسی[1]، یکی از چالش‌های مهم این حوزه طراحی استراتژی‌های کارآمد برای تخلیه وظایف بوده است.

علاوه بر این، با رشد روز افزون صنعت تلفن همراه و اینترنت اشیا، انواع زیادی از کاربردهای نرم‌افزاری جدید با نیازمندی‌های پردازشی متفاوت در سطح شبکه به وجود آمده است. بنابراین یک ویژگی مهم در طراحی استراتژی تخلیه وظایف در رایانش لبه‌ای، در نظر گرفتن ناهمگونی کاربردها از نظر میزان منابع مورد نیاز است.

در این مقاله روشی برای بدست آوردن استراتژی تخلیه وظایف با تاخیر کمینه تحت محدودیت توان مصرفی ارائه شده است. روش پیشنهادی شامل دو قسمت می‌باشد. در قسمت اول، سیستم تخلیه وظایف با کمک زنجیره مارکوف گسسته-زمان مدل‌سازی می‌شود و در قسمت دوم، با استفاده از الگوریتمی مبتنی بر برنامه‌ریزی خطی[2] استراتژی تخلیه بهینه برای مدل ساخته شده محاسبه می‌شود.

علاوه بر تشریح و حل مسئله به صورت تئوری، ساختار نرم‌افزاری جدیدی در زبان Kotlin ارائه می‌شود که می‌توان با استفاده از آن استراتژی بهینه را برای سیستم مورد نظر بدست آورد و عملکرد آن را با کمک شبیه‌سازی بررسی کرد. مقاله فعلی گسترشی بر پژوهش [۱] است و از روشی مشابه با روش ارائه شده در این پژوهش استفاده می‌کند.

واژگان کلیدی: تخلیه وظیفه، رایانش لبه‌ای، زنجیره مارکوف، برنامه‌ریزی خطی، رایانش ابری

[1] Multi-access Edge Computing

[2] Linear Programming

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 تخلیه 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 اشیا 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. معماری رایانش لبه‌ای و پیاده‌سازی‌های 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]

یک امر مهم در پیاده‌سازی کارآمد رایانش لبه‌ای، طراحی استراتژی‌های تخلیه وظایف هوشمند و موثر است. این استراتژی‌ها نحوه تخصیص منابع توسط دستگاه کاربر[1] را مشخص می‌کنند و این امکان را به دستگاه کاربر می‌دهند تا درباره تخلیه یا عدم تخلیه وظایف محاسباتی در طول زمان تصمیم بگیرد.

در این پژوهش روشی برای بدست آوردن استراتژی تخلیه با تاخیر کمینه در محیط رایانش لبه‌ای ارائه خواهیم داد که مبتنی بر زنجیره مارکوف گسسته-زمان و برنامه‌ریزی خطی می‌باشد. روش پیشنهادی 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 اشیا بوده است. به طور مثال در بسیاری از پژوهش‌های حوزه تخلیه وظیفه در اینترنت اشیا، 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 این مقاله به پنج فصل تقسیم شده است. در فصل ۲ پژوهش‌های مرتبط انجام شده را مرور می‌کنیم. در فصل ۳ به شرح مسئله تخلیه 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[2] 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[3] مقایسه کنند. در بخش دوم از فصل ۴ با استفاده از Kompute به طور جامع 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.

[1] User Equipment

[2] Framework

[3] Baseline

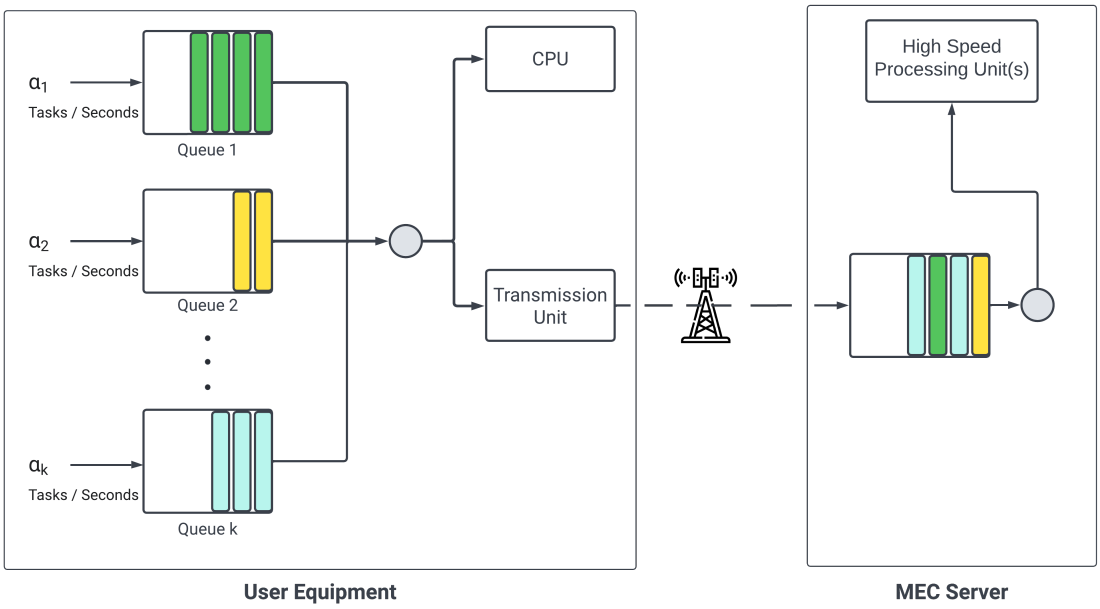
فصل ۲

مروری بر ادبیات و کارهای انجام شده

فصل ۳

شرح مسئله

در این مقاله 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 این مقاله 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 تخلیه 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 تخلیه وظیفه با چند نوع وظیفه قابل استفاده نیست و نیاز به تغییر دارد. ما در این مقاله برای تعیین حالت دستگاه کاربر از یک چندتایی[۶] به طول $k+4$ مطابق با رابطه ۳-۱ 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.

$$\tau[t] = (q_1[t], q_2[t], \ldots, q_k[t], c_R[t], c_L[t], T_R[t], T_L[t]) \qquad (۱-۳)$$

رابطه ۳-۲ با تعریف شروط مختلف فضای حالت مسئله را توصیف می‌کند. **(نکته: در رابطه ۳-۲ و سراسر این مقاله منظور از $\tau\{X\}$ مقدار متغیر X در حالت τ است.)**

$$\begin{aligned}
&\forall \tau \in S, i \in \{1,2,\ldots,k\} \quad 0 \leqslant \tau\{q_i\} \leqslant Q \\
&\forall \tau \in S \quad \tau\{T_L\}, \tau\{T_R\} \in \{0,1,2,\ldots,k\} \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_R\} = 0\} \quad \tau\{C_R\} = 0 \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_R\} \neq 0\} \quad 1 \leqslant \tau\{C_R\} \leqslant M_{\tau\{T_R\}} \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_L\} = 0\} \quad \tau\{C_L\} = 0 \\
&\forall \tau \in \{\tau' \in S \mid \tau'\{T_L\} \neq 0\} \quad 1 \leqslant \tau\{C_L\} \leqslant L_{\tau\{T_L\}} - 1
\end{aligned} \qquad (۲-۳)$$

[6] Tuple

۳-۳ مدل زمان

وضعیت سیستم تخلیه 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. [۵]

۳-۴ مدل کانال بیسیم

در این مقاله 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 ۳-۳ (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.

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

[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 $A[t]$ 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.

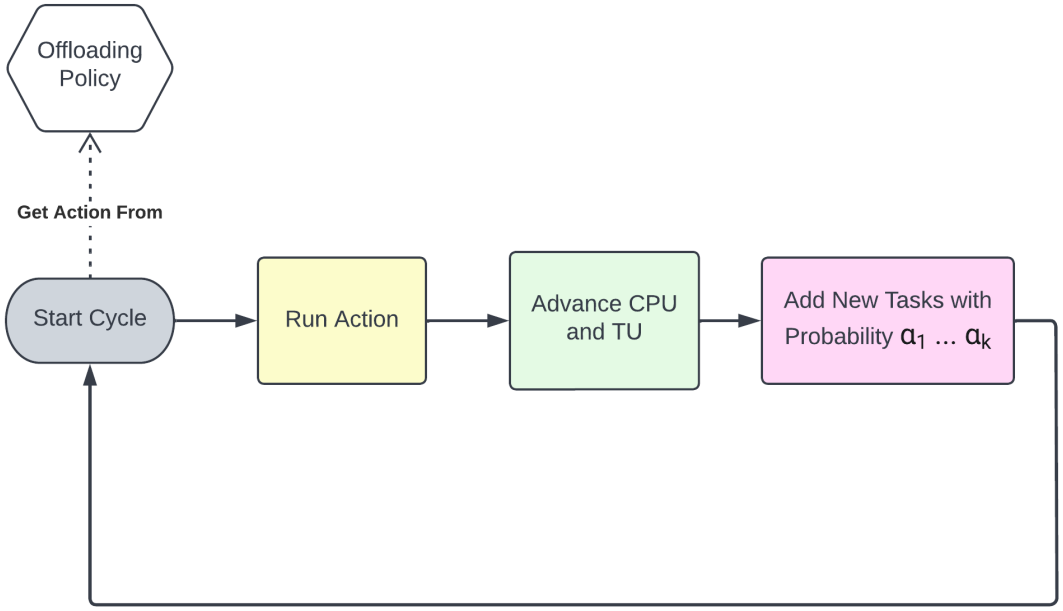
۳-۶ استراتژی تخلیه

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 تخلیه وظایف

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 باید برای هر استراتژی تخلیه تصادفی برقرار باشد.

$$\sum_{\tau \in S} \sum_{a \in A} g_{\tau}^{a} = 1 \tag{۴-۱}$$

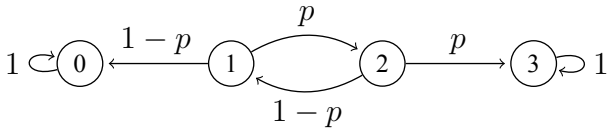
۴-۲ مدل زنجیره مارکوف دستگاه کاربر

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 تخلیه 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

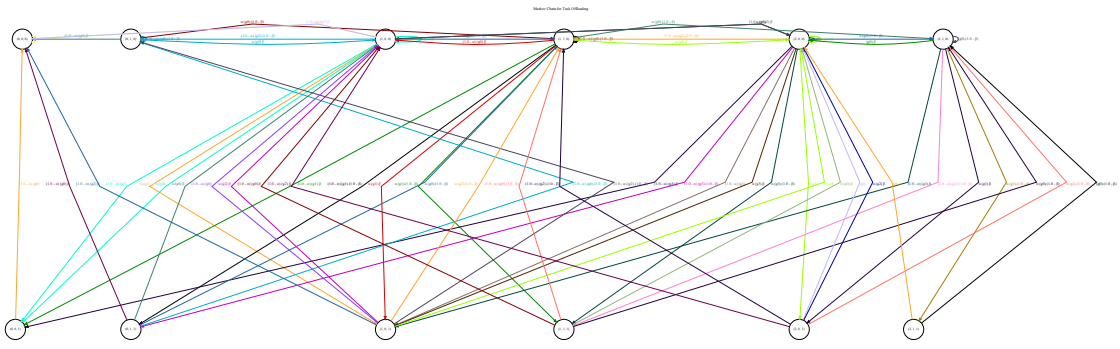
تعریف ۲.۴. زنجیره مارکوف گسسته زمان $X(t)$ را **همگن-زمان** می‌گوییم اگر شرط زیر همواره برقرار باشد:

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

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. ماتریس انتقال را می‌توان به صورت یک گراف جهت‌دار نیز توصیف کرد به طوری که درایه $p_{i,j}$ در ماتریس معادل یک یال جهت‌دار از راس i به راس j با وزن $p_{i,j}$ است.

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.

ماتریس انتقال را می‌توان به ازای یک استراتژی تخلیه داده شده و پارامترهای سیستمی مانند $\alpha_1,\cdots,\alpha_k$ بدست آورد. در شکل ۴-۲ گراف جهت‌دار معادل زنجیره مارکوف برای یک سیستم با یک صف وظیفه و $Q=2$ و تعداد ۲ قسمت به ازای هر وظیفه و تعداد ۱ بسته به ازای هر وظیفه رسم شده است.[۲]



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

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

به منظور محاسبه معیارهایی مانند توان مصرفی میانگین و تاخیر سرویس میانگین لازم است که بتوانیم درباره وضعیت سیستم تخلیه وظیفه در طولانی‌مدت استنتاج کنیم. در همین راستا مفهوم توزیع پایدار را 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:

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

یک سوالی که ممکن است بوجود بیاید این است که آیا هر زنجیره مارکوف گسسته‌زمان 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 تخلیه 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 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.

$$\begin{cases} \sum_{\boldsymbol{\tau}' \in \mathcal{S}} \chi_{\boldsymbol{\tau}', \boldsymbol{\tau}} \pi_{\boldsymbol{\tau}'} = \pi_{\boldsymbol{\tau}}, \forall \boldsymbol{\tau} \in \mathcal{S} \\ \sum_{\boldsymbol{\tau} \in \mathcal{S}} \pi_{\boldsymbol{\tau}} = 1 \end{cases} \tag{۴-۲}$$

۴-۳ 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 بدست می‌آید.

$$\theta_i = \frac{\alpha_i}{\sum_{j=1}^{k} \alpha_j} \tag{۴-۳}$$

[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 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.

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

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 ۴-۵ بدست می‌آید.

$$t_{tx}^i = M_i \sum_{j=1}^{\infty} j(1-\beta)^{(j-1)}\beta \tag{۵-۴}$$

به یاد داریم که مقدار تاخیر در صورت پردازش محلی برای وظایف نوع i برابر L_i می‌باشد. تاخیر اجرا در صورت تخلیه 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.

$$t_c^i = t_{tx}^i + C_i + t_{rx} \tag{۶-۴}$$

در نتیجه می‌توان تاخیر اجرای میانگین وظایف نوع i را نیز مطابق رابطه ۴-۷ بیان کرد.

$$t_p^i = \eta_i L_i + (1-\eta_i) t_c^i \tag{۷-۴}$$

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 بدست می‌آید.

$$\eta_i = \frac{\sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_3^i\cup\mathcal{S}_5^i} \pi_{\boldsymbol{\tau}} g_{\boldsymbol{\tau}}^a}{\sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_2^i\cup\mathcal{S}_3^i\cup\mathcal{S}_4^i} \pi_{\boldsymbol{\tau}} g_{\boldsymbol{\tau}}^a + 2\sum_{\boldsymbol{\tau},\boldsymbol{a}\in S_5^i} \pi_{\boldsymbol{\tau}} g_{\boldsymbol{\tau}}^a} \tag{۴-۸}$$

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

(۴-۹)

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

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

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

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

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

در رابطه فوق تابع $type(a)$ نوع کنش را مشخص می‌کند و یکی از چهار نوع بیان شده در بخش ۳-۵ می‌باشد. توابع $locType(a)$ و $offloadType(a)$ 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 محاسبه کنیم. رابطه بدست آمده برای $\bar{T}$ همچنین مشخص کننده تابع هدف در مسئله پیدا کردن استراتژی تخلیه بهینه می‌باشد.

$$\bar{T} = \sum_{i=1}^{k} \theta_i \left(t_q^i + t_p^i\right) \tag{۴-۱۰}$$

۴-۴ توان مصرفی میانگین

اگر پارامتر μ_τ^{loc} و μ_τ^{tx} را که به ترتیب به عنوان احتمال فعالیت پردازنده در حالت τ و احتمال فعالیت واحد ارسال در حالت τ تعریف کنیم، آنگاه توان مصرفی میانگین طبق رابطه زیر بدست می‌آید:

$$\bar{P} = \sum_{\boldsymbol{\tau} \in \mathcal{S}} \pi_{\boldsymbol{\tau}} \left(\mu_{\boldsymbol{\tau}}^{loc} P_{loc} + \mu_{\boldsymbol{\tau}}^{tx} P_{tx} \right) \tag{۱۱-۴}$$

۴-۵ استراتژی تخلیه 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:

$$\begin{aligned} \mathcal{P}_1 : \min_{\{g_\tau^a\}} \bar{T} &= \left(\sum_{i=1}^{k} \frac{1}{\alpha_i} \sum_{\tau \in S} \tau\{q_i\} \cdot \pi_\tau\right) + T_p^0 \\ \text{s.t.} \quad & \begin{cases} \bar{P} \le \bar{P}_{\max} \\ \sum_{\boldsymbol{\tau}' \in \mathcal{S}} \chi_{\boldsymbol{\tau}', \boldsymbol{\tau}} \pi_{\boldsymbol{\tau}'} = \pi_{\boldsymbol{\tau}}, \boldsymbol{\tau} \in \mathcal{S}, \\ \sum_{\boldsymbol{\tau} \in S} \pi_{\boldsymbol{\tau}} = 1, \\ \sum_{\alpha \in A} g_\tau^\alpha = 1, \forall \tau \in S \\ g_\tau^a \ge 0, \forall \tau \in S,\ a \in A \end{cases} \end{aligned} \tag{۱۲-۴}$$

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:

$$T_p^0 = \sum_{i=1}^{k} (\eta_i L_i + (1 - \eta_i) t_c^i) \tag{۱۳-۴}$$

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:

$$\mathcal{P}_2 : \min_{\boldsymbol{x},\eta} \bar{T} = (\sum_{i=1}^{k} \frac{1}{\alpha_i} \sum_{\tau\in\mathcal{S}} \sum_{a\in A} \tau\{q_i\} \cdot x_\tau^a) + T_p^0$$
$$\text{s.t.} \begin{cases} \nu_{loc}(\boldsymbol{x}) P_{loc} + \beta \nu_{tx}(\boldsymbol{x}) P_{tx} \le \bar{P}_{\max} \\ \Gamma(\boldsymbol{x}, \eta_i) =, \forall i \in \{1, \cdots, k\} \\ F_\tau(\boldsymbol{x}) = 0, \forall \tau = (i, m, n) \in \mathcal{S} \\ \sum_{\tau\in\mathcal{S}} \sum_{a\in A} = 1 \\ \eta_i \in [0,1], \forall i \in \{1, \cdots, k\} \\ x_\tau^a \ge 0, \forall \tau \in S, a \in A \end{cases} \qquad (۱۴-۴)$$

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 ۴-۸ می‌باشد و به صورت زیر محاسبه می‌شود:

$$\Gamma(x, \eta) = \eta \sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_2^i\cup\mathcal{S}_3^i\cup\mathcal{S}_4^i} x_\tau^a + 2\eta \sum_{\boldsymbol{\tau},\boldsymbol{a}\in S_5^i} x_\tau^a - \eta \sum_{\boldsymbol{\tau},\boldsymbol{a}\in\mathcal{S}_1^i\cup\mathcal{S}_3^i\cup\mathcal{S}_5^i} x_\tau^a \qquad (۱۵-۴)$$

[4] Occupation Measure

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

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

$$F_\tau(\boldsymbol{x}) = \sum_{\tau' \in \mathcal{S}} \sum_{a \in A} \tilde{\chi}_{\tau',\tau,a} x^a_{\tau'} - \sum_{a \in A} x^a_\tau \qquad (۱۶-۴)$$

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.

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 $\{x^a_\tau\}$ می‌توان استراتژی بهینه را طبق رابطه زیر بدست آورد:

$$g^{a*}_\tau = \frac{x^{a*}_\tau}{\sum_{a \in A} x^{a*}_\tau}, \forall \tau \in \mathcal{S}, a \in A \qquad (۱۷-۴)$$

بنابراین جهت یافتن استراتژی بهینه برای یک سیستم تخلیه 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.

الگوریتم ۱.۴ الگوریتم جستجوی استراتژی تخلیه وظیفه بهینه

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

۴-۶ 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 بخش پیش رو ارائه خواهد شد پیاده‌سازی شده اند.

۴-۶-۱ کاهش تعداد متغیرها

در مسئله بهینه‌سازی $\mathcal{P}_2$ تعداد $|S| \cdot |A|$ متغیر وجود دارد. این مقدار برای تعداد صف‌های کم (برای مثال $k \leq 4$) 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.

۴-۶-۲ 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/گره پردازشی خاص اختصاص داد. در شبیه‌سازی سناریو وظایف سبک و سنگین (رجوع شود به ۵-۲-۱) 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] داشت.

[6] Scaling

[7] Thread

[8] Single-thread

فصل ۵

آزمایش و نتایج

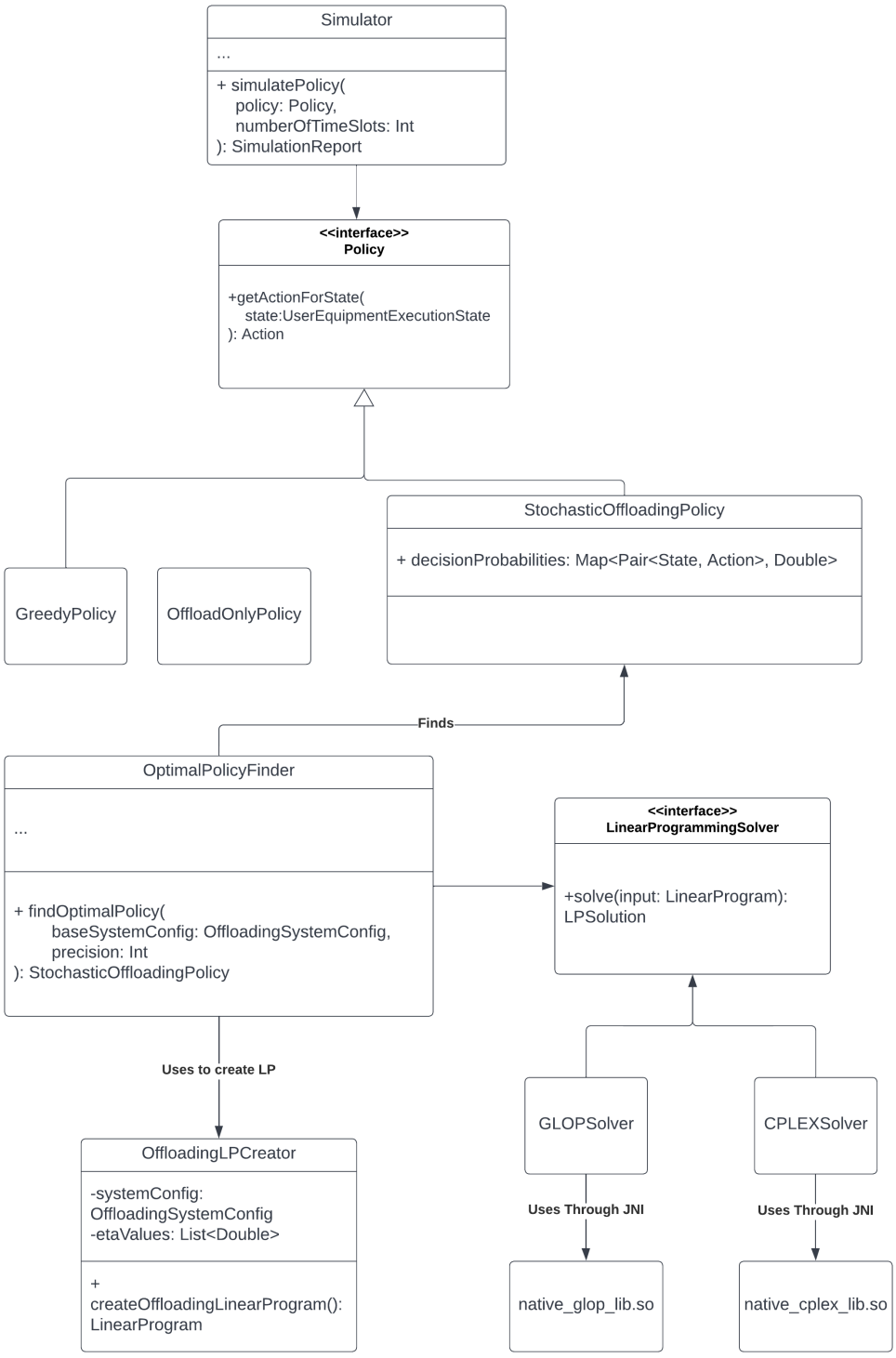
۵–۱ ساختار نرم‌افزاری Kompute

به منظور تست الگوریتمی که در بخش پیشین ارائه گردید، یک ساختار نرم‌افزاری (فریم‌ورک) با نام Kompute در زبان Kotlin تعبیه شده است که مخزن آن در لینک زیر در دسترس می‌باشد:

- https://github.com/dalisyron/Kompute

با استفاده از این فریم‌ورک می‌توان الگوریتم جستجوی استراتژی تخلیه بهینه را به ازای پارامترهای محیطی مختلف اجرا کرد و عملکرد استراتژی بدست آمده از الگوریتم را با کمک شبیه‌سازی با سایر استراتژی‌ها مقایسه کرد. این فریم‌ورک طبق یافته‌های ما اولین پیاده‌سازی متن‌باز در زمینه استراتژی تخلیه وظایف ناهمگون است.

معماری کلی این فریم‌ورک در قالب یک کلاس دیاگرام در صفحه بعد آورده شده است.



شکل ۵-۱: کلاس دیاگرام فریم‌ورک Kompute

۵-۱-۱ ساخت و حل یک مسئله تخلیه وظیفه نمونه در Kompute

در کد نمونه زیر مسئله تخلیه وظیفه برای محیط رایانش لبه‌ای با دو صف[۱] حل شده است.

```
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(0.4, 0.9),
            beta = 0.90,
            etaConfig = null,
            pTx = 1.0,
            pLocal = 0.8,
            pMax = 1.7
         )
      ),
      environmentParameters = EnvironmentParameters(
         nCloud = listOf(1, 1),
         tRx = 0.5,
      )
   )

   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)
}
```

[۱] شرایط بر اساس تقسیم‌بندی وظایف به Heavy و Light در اینترنت اشیا

۵-۲ نتایج شبیه‌سازی

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

۱. استراتژی «فقط تخلیه»[۲] که همه‌ی وظایف را تخلیه می‌کند

۲. استراتژی «حریصانه، تخلیه 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، تخلیه را انتخاب می‌کند.

۳. استراتژی «حریصانه، محلی اول»[۴] 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»[۵]

[2] Offload Only

[3] Greedy (Offload First)

[4] Greedy (Local First)

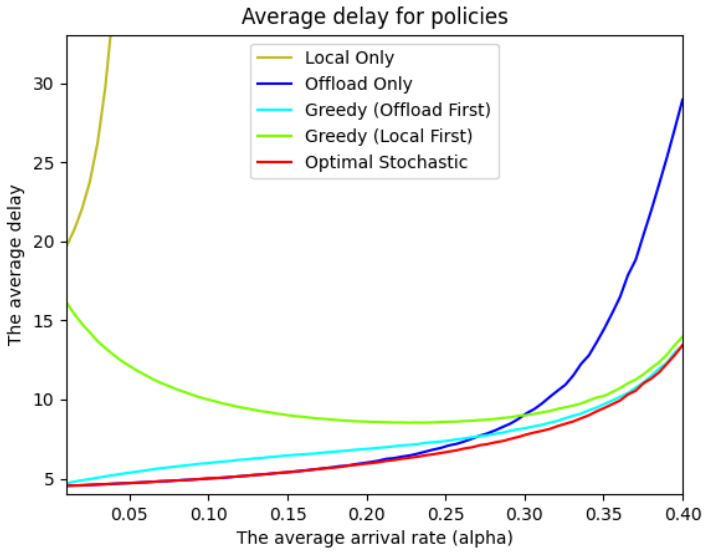
[5] Local Only

۵-۲-۱ 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 ۵-۱ خلاصه شده اند. نتیجه این آزمایش در شکل ۵-۲ مشاهده می‌شود.

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 ۵-۱: پارامترهای محیط رایانش لبه‌ای در سناریو تک صف

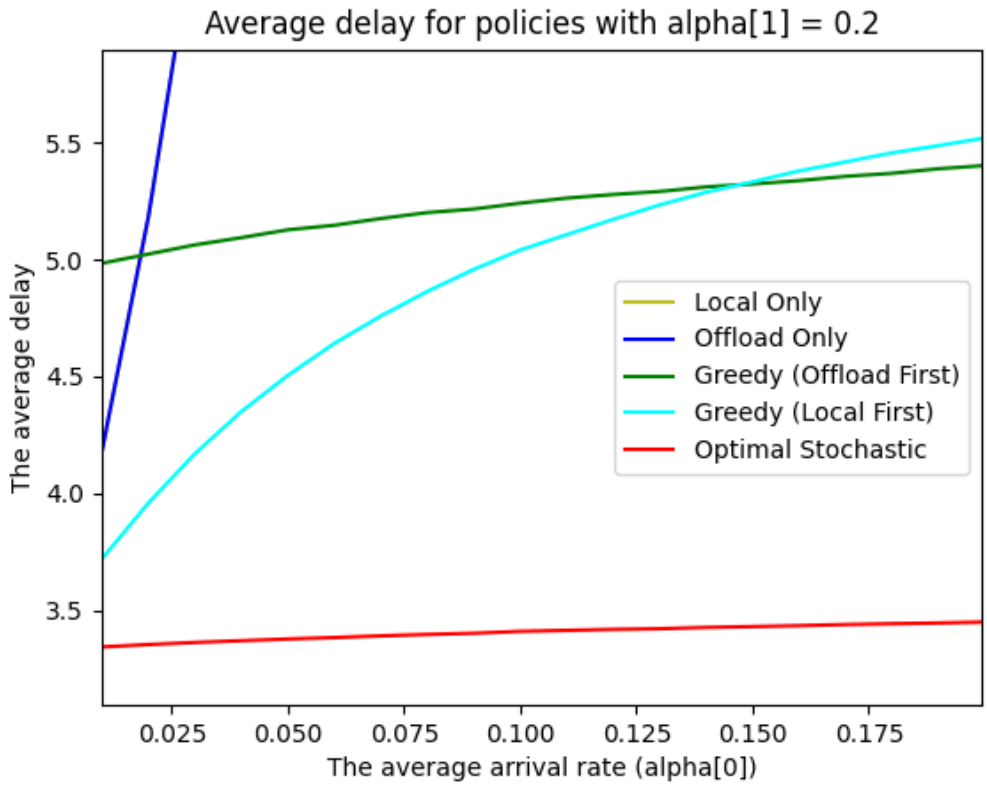


شکل ۵-۲: تاخیر سرویس به ازای نرخ ورود در حالت تک صف

همانطور که مشاهده می‌شود استراتژی تخلیه تصادفی یافت شده از تمام الگوریتم‌های پایه بهتر عمل می‌کند و شکل منحنی‌های نمودار با [۱] مطابقت دارد.

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 ۵-۲ به طور خلاصه آمده است.



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

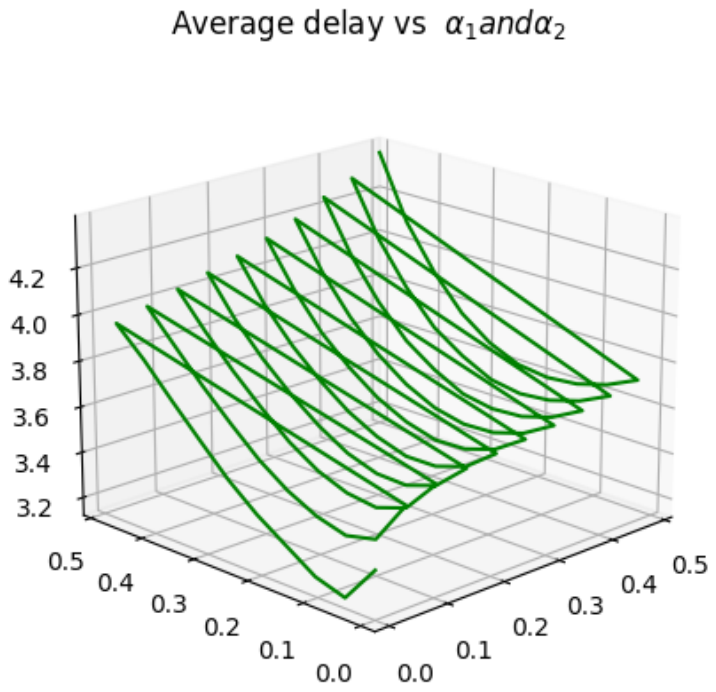
جدول ۵-۲: پارامترهای محیط رایانش لبه‌ای در سناریو دو صف با یک صف ثابت

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 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.

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 ۵-۳ آمده است. همانطور که مشاهده می‌شود استراتژی بهینه در بازه $\alpha_1, \alpha_2 \in [0, 0.4]$ عملکرد قابل قبول دارد.



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 ۵-۳: پارامترهای محیط رایانش لبه‌ای در سناریو دو صف متغیر

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 ۵-۵ 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 ۵-۴ 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 ۵-۴: 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 ۵-۵: پارامترهای محیط رایانش لبه‌ای در سناریو سه صف

فصل ۶

جمع‌بندی و پیشنهادها

در این پژوهش روشی برای بدست آوردن استراتژی تخلیه وظیفه با تاخیر کمینه در شرایط حضور چندین نوع وظیفه در محیط رایانش لبه‌ای معرفی شد. عملکرد این الگوریتم نیز با کمک شبیه‌سازی بررسی شد. در طول انجام این پژوهش ایده‌های مختلفی برای بهبود روش ارائه شده به ذهن ما رسید که برخی از آنها مانند بهینه‌سازی‌های معرفی شده در بخش ۴-۶ پیاده‌سازی شدند. اما برخی از این ایده‌ها به مرحله پیاده‌سازی نرسیدند و پژوهش درباره آنها امکان بهبود روش فعلی را فراهم خواهد کرد.

یکی از این موارد کاهش تعداد متغیرهای مسئله برنامه‌ریزی خطی $\mathcal{P}_2$ (رابطه ۴-۱۴) با استفاده از حذف «تک کنش» ها می‌باشد. پیشتر در بخش ۴-۶-۱ 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 امکان وجود دارد که متغیرهایی که متناظر با تنها کنش ممکن در یک حالت هستند را از الگوریتم حذف کرد، زیرا مقدار این متغیرها در تعیین استراتژی بهینه نقشی ندارد چون احتمال انتخاب آنها همواره ۱ (قطعی) می‌باشد. با این حال حذف این متغیرها بر خلاف بهینه‌سازی ۴-۶-۱ ساختار زنجیره مارکوف را دگرگون خواهد، بنابراین احتمالا نیازمند تغییر توابع انتقال و/یا تغییر شروط رابطه ۴-۱۴ خواهد شد.

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
   )
}
```

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

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 smartphone and IoT industry rapidly growing, many new types of applications have been added 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 includes two parts. 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 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

Stochastic Computation Offloading Policy for Heterogeneous Tasks

Bachelor of Science Thesis in Computer Engineering

By:

Mohammadmobin Dariushhamedani

Supervisor:

Dr. Reza Entezari-Maleki

June 2022