بسمه تعالی

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

گزارش‌کار فاز دوم پروژه

سامانه‌های بی‌درنگ

مکانیزم بار سپاری پویا برای محیط VEC

ثنا بابایان

(۴۰۱۱۰۵۶۸۹)

سید علی طیب

(۴۰۰۱۰۱۵۲۶)

صفری

بهار ۱۴۰۴

مقدمه:

در این پروژه، هدف پیاده‌سازی یک **مکانیزم بارسپاری پویا** برای تسک‌های با اهمیت‌های مختلف در محیط VEC است. این مکانیزم به‌صورت ویژه به مدیریت بهینه‌ی منابع پردازشی در دیوایس‌های دارای محدودیت توان پردازشی می‌پردازد.

در محیط VEC، تسک‌ها بر اساس اهمیت به سه تا چهار سطح تقسیم می‌شوند. در این میان، **تسک‌های حیاتی** مهم‌ترین نوع تسک‌ها هستند؛ زیرا اجرای نادرست آن‌ها می‌تواند منجر به رخدادهای فاجعه‌بار شود. برای نمونه، در یک خودروی خودران، عدم شناسایی به‌موقع یک شیء ممکن است منجر به تصادف و آسیب‌های جانی گردد. به همین دلیل، این تسک‌ها باید حتماً به‌صورت محلی (on-device) پردازش شوند تا از تأخیرهای احتمالی ناشی از ارسال داده به سرورهای لبه جلوگیری شود.

تخمین ما نشان می‌دهد که در بدترین حالت، باید حدود ۶۰٪ از توان پردازشی دیوایس به این تسک‌های حیاتی اختصاص یابد. بنابراین، این منابع از سایر پردازش‌ها جدا شده و صرفاً برای تسک‌های حساس رزرو می‌شوند. ظرفیت باقی‌مانده می‌تواند برای اجرای تسک‌های با اهمیت متوسط یا پایین‌تر مورد استفاده قرار گیرد.

چالش اصلی این پروژه اینجاست: در بسیاری از مواقع، تسک‌های حیاتی نیاز پردازشی کمتری نسبت به ۶۰٪ رزرو شده دارند. این وضعیت معمولاً در شرایط سرعت بالا رخ می‌دهد که تعداد تسک‌های مهم افزایش می‌یابد، اما در شرایط عادی، بخشی از منابع بدون استفاده باقی می‌ماند.

ایده‌ی اصلی پروژه این است که از این ظرفیت بلااستفاده برای اجرای سایر تسک‌ها استفاده کنیم، اما اولویت اصلی را همچنان برای تسک‌های حیاتی حفظ کنیم. به این معنا که اگر هنگام اجرای یک تسک کم‌اهمیت، یک تسک حیاتی وارد سیستم شود و منابع لازم برای اجرای آن در دسترس نباشد، فوراً تسک کم‌اهمیت متوقف شده و منابع در اختیار تسک حیاتی قرار می‌گیرد. تسک متوقف‌شده، اگر محدودیت‌های زمانی اجازه دهد، مجدداً در صف پردازش قرار خواهد گرفت؛ در غیر این صورت، به عنوان تسک شکست‌خورده کنار گذاشته می‌شود.

این راهکار، بدون به خطر انداختن اجرای تسک‌های حیاتی، باعث بهره‌وری بیشتر از منابع پردازشی خواهد شد.

هدف پروژه پیاده‌سازی این مکانیزم و طراحی و توسعه‌ی یک **الگوریتم متاهیوریستیک قدرتمند** است. این الگوریتم با مشورت دستیار آموزشی و استاد راهنما انتخاب و طراحی خواهد شد تا بهترین مکانیزم آفلودینگ ممکن را ارائه دهد. مکانیزم پیشنهادی باید با در نظر گرفتن ظرفیت پردازشی پویا در دیوایس، به تخصیص بهینه‌ی منابع بین تسک‌های مختلف کمک کند و بهره‌وری کلی را افزایش دهد.

مطالعه و انتخاب الگوریتم تخصیص (ACO)

پس از تعریف اولیه‌ی مسئله، مقاله‌ی AVE: Autonomous Vehicular Edge Computing Framework with ACO-Based Scheduling را مطالعه کردیم که از بسیاری جهات با پروژه‌ی ما شباهت داشت. این مقاله به بررسی یک چارچوب محاسباتی در لبه برای وسایل نقلیه‌ی خودران می‌پردازد و از الگوریتم کلونی مورچه‌ها (Ant Colony Optimization - ACO) برای زمان‌بندی و تخصیص بهینه‌ی منابع استفاده می‌کند.

مطالعه‌ی این مقاله ما را به این نتیجه رساند که استفاده از الگوریتم ACO می‌تواند در پروژه‌ی ما نیز مؤثر باشد. به همین دلیل، تصمیم گرفتیم که نسخه‌ای اولیه از این الگوریتم را پیاده‌سازی کنیم.

الگوریتم ACO یک الگوریتم متاهیوریستیک الهام‌گرفته از رفتار مورچه‌ها در طبیعت است. مورچه‌ها برای یافتن کوتاه‌ترین مسیر بین لانه و منبع غذا از یک سازوکار شیمیایی به‌نام فرومون استفاده می‌کنند. الگوریتم ACO نیز با استفاده از همین ایده، مجموعه‌ای از مسیرهای ممکن را شبیه‌سازی کرده و با گذشت زمان و به‌روزرسانی میزان فرومون، مسیر بهینه را شناسایی می‌کند. این الگوریتم معمولاً در مسائل بهینه‌سازی گسسته و زمان‌بندی (مانند مسئله‌ی ما) کاربرد دارد.

برای پیاده‌سازی اولیه‌ی الگوریتم ACO، یک پوشه به نام aco به دایرکتوری controllers پروژه اضافه کردیم و یک کلاس با نام AcoZoneManager طراحی کردیم که ساختار اولیه‌ی آن به‌شکل زیر بود:

توضیح تابع assign_task

تابع assign_task در کلاس AcoZoneManager مسئول تخصیص یک تسک (وظیفه) به یکی از نودهای fog در دسترس است. این تابع به صورت پویا تصمیم می‌گیرد که تسک به کدام نود اختصاص یابد، با توجه به نوع تسک (بحرانی یا معمولی)، جدول فرومون‌ها، و ارزیابی اکتشافی (heuristic) مربوط به هر نود.

الگوریتم در دو حالت کلی رفتار می‌کند:

1. تسک‌های بحرانی (CRUCIAL)

در صورتی که تسک دارای اولویت بحرانی باشد، تلاش می‌شود که آن را مستقیماً به ۶۰٪ ظرفیت یکی از نودهای موبایل اختصاص دهیم. در اینجا، ۶۰٪ ظرفیت برای تسک‌های مهم رزرو شده است. فرآیند به این شکل است:

در لیست نودهای ممکن، به دنبال یک نود موبایل می‌گردیم که فضای کافی در ۶۰٪ ظرفیت خود داشته باشد.

در صورت موفقیت، تسک به آن نود اختصاص می‌یابد و مقدار فرومون بین آن تسک و آن نود به‌روزرسانی می‌شود (افزایش ۰.۱). اگر هیچ نودی فضای آزاد نداشته باشد، تلاش می‌کنیم برخی از تسک‌های کم‌اولویت‌تر را پیش‌دستی (preempt) کنیم تا جا برای تسک بحرانی باز شود. تسک‌هایی که از روی نود حذف شده‌اند (پیش‌دستی‌شده‌ها)، در لیست deferred_tasks قرار می‌گیرند تا در آینده مجدداً زمان‌بندی شوند. اگر حتی با پیش‌دستی هم تسک قابل تخصیص نباشد، یک خطای RuntimeError ایجاد می‌شود.

2. تسک‌های معمولی (LOW_PRIORITY یا HIGH_PRIORITY)

در این حالت، از الگوریتم ACO برای انتخاب نود استفاده می‌شود:

ابتدا برای هر نود یک نمره (score) محاسبه می‌شود که برابر است با حاصل‌ضرب مقدار فرومون بین آن تسک و آن نود در مقدار اکتشافی (heuristic). این مقدار اکتشافی می‌تواند بر اساس ویژگی‌هایی مانند بار فعلی نود، تأخیر شبکه، و غیره محاسبه شده باشد. سپس بر اساس یک انتخاب تصادفی وزن‌دار (weighted random selection)، یکی از نودها انتخاب می‌شود. این روش مشابه رفتار مورچه‌هاست که با احتمال بالاتر به مسیرهایی با فرومون بیشتر و مسیرهای کوتاه‌تر می‌روند. اگر به دلایلی مجموع نمرات صفر بود، یکی از نودها به‌صورت تصادفی انتخاب می‌شود (fallback). پس از انتخاب نود، تسک به آن اختصاص داده می‌شود (در صورت امکان، در ۶۰٪ نود موبایل برای تسک کم‌اولویت). مقدار فرومون بین تسک و نود انتخاب‌شده به میزان ۰.۱ افزایش می‌یابد. این تابع ترکیبی از یک سیاست سخت‌گیرانه برای تسک‌های بحرانی و یک استراتژی مبتنی بر ACO برای تسک‌های غیر بحرانی را پیاده‌سازی می‌کند. به این ترتیب، هم نیازهای فوری تسک‌های حساس به زمان تأمین می‌شود و هم در بلندمدت، بهینه‌سازی مسیرهای تخصیص بر اساس یادگیری تدریجی صورت می‌گیرد.

توضیح منطق ۶۰٪ در models/nodes/base.py

در این بخش از کد، منطق «۶۰٪» به منظور مدیریت بهینه منابع و اولویت‌بندی تسک‌ها در نودهای موبایل (به خصوص MobileFogNode) پیاده‌سازی شده است. ایده اصلی این است که برای برخی تسک‌ها، به ویژه تسک‌های با اولویت پایین‌تر یا تسک‌هایی که منابع زیادی مصرف می‌کنند، تنها ۶۰٪ از ظرفیت کل نود برای پردازش آن‌ها اختصاص داده شود. این منطق به چند دلیل اهمیت دارد:

مدیریت منابع محدود: نودهای موبایل معمولا منابع محدودی دارند (مثل توان پردازشی و انرژی). اختصاص دادن فقط ۶۰٪ از منابع به یک تسک باعث می‌شود منابع برای سایر تسک‌ها و درخواست‌ها هم باقی بماند.

حمایت از تسک‌های حیاتی: در صورت وجود تسک‌های با اولویت بالاتر (مثلاً تسک‌های حیاتی)، اگر ظرفیت باقی‌مانده برای پردازش تسک حیاتی کافی نباشد، تسک‌های با اولویت پایین‌تر می‌توانند به صورت موقت پیش‌دستی (Preemption) شوند و منابع آزاد شوند.

پیش‌دستی هوشمندانه: کد قابلیت پیش‌دستی تسک‌های کم‌اولویت را دارد تا بتواند ظرفیت مورد نیاز برای تسک‌های مهم‌تر را فراهم کند، به این ترتیب تضمین می‌شود که تسک‌های حیاتی بدون مشکل اجرا شوند. در تابع can_offload_task شرایط مختلفی برای بررسی قابلیت انجام تسک در نود وجود دارد، از جمله محدودیت صف تسک‌ها، توان مصرفی، اولویت تسک و فاصله مکانی. بخشی از این منطق مرتبط با بررسی توان باقی‌مانده و تطابق آن با «۶۰٪» منابع است. تابع assign_task با توجه به اولویت تسک، ممکن است تسک را در حالت «۶۰٪» تخصیص دهد، یا اگر تسک حیاتی است و منابع کافی نیست، تسک‌های کم‌اولویت‌تر را پیش‌دستی کند تا فضای لازم فراهم شود. در این فرآیند، تسک‌های پیش‌دستی‌شده دوباره در صف قرار می‌گیرند تا در زمان مناسب پردازش شوند. تابع execute_tasks مسئول اجرای تسک‌های جاری و مدیریت زمان‌بندی و آزادسازی منابع است، که منطق «۶۰٪» را با توجه به توان باقی‌مانده و اولویت تسک‌ها رعایت می‌کند.

کدهای پروژه:

کدهای پروژه را می‌توانید در مخزن و شاخه زیر مشاهده کنید. کد اولیه این مخزن از طرف دستیار آموزشی در اختیار ما قرار داده شده است و قرار است برای انجام این پروژه، کد اولیه را تکمیل کنیم:

https://github.com/AbbasShabrang/VFC_BASE/tree/Tayyeb