دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی

برون‌سپاری وظایف پردازنده در سطح سیستم‌عامل با استفاده از روش مهاجرت وظایف بین دو کامپیوتر

نگارنده

محمدعلی مقیمی

استاد راهنما

دکتر حمیدرضا زرندی

مرداد ۱۴۰۳

صفحه فرم ارزیابی و تصویب پایان‌نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه (هر سه مقطع تحصیلی) باید تصویر فرم ارزیابی یا تأیید و تصویب پایان‌نامه/رساله موسوم به فرم کمیته دفاع برای مقاطع کارشناسی‌ارشد و دکتری و تصویر فرم تصویب برای مقطع کارشناسی، موجود در پرونده آموزشی را قرار دهند.

نکات مهم:

- ✓ نگارش پایان‌نامه/رساله باید به زبان فارسی و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان‌نامه‌های دانشگاه صنعتی امیرکبیر باشد. (دستورالعمل و راهنمای حاضر)
- ✓ تحویل پایان‌نامه به زبان انگلیسی، برای دانشجویان بین‌الملل با شرایط دستورالعمل حاضر بلامانع است و داشتن صفحه عنوان فارسی به همراه چکیده مبسوط فارسی، ۳۰صفحه برای پایان‌نامه کارشناسی‌ارشد و ۵۰صفحه برای رساله دکتری در ابتدای آن الزامی است.
- ✓ دریافت پایان‌نامه کارشناسی، کارشناسی‌ارشد و رساله دکتری، به‌صورت نسخه الکترونیکی مطابق راهنمای وبسایت و دستورالعمل حاضر می‌باشد.
- ✓ درصورتی که یک عنوان پایان‌نامه دارای دو نویسنده است، فقط یکبار فایل و فرم اطلاعات آن با ذکر هر دو نویسنده بارگذاری و تکمیل گردد.
- ✓ با توجه به اینکه در Word ۲۰۱۶یا بالاتر، احتمال تغییر ترتیب ذکر زیر فصل‌ها وجود دارد لطفا در انتها به شماره‌دهی زیر فصل‌ها توجه نمایید که به‌صورت صحیح باشد. از راست به چپ: شماره فصل-زیرفصل۱-زیرفصل۲-زیرفصل۳ و ...

به نام خدا

تاریخ:

تعهدنامه اصالت اثر

اینجانب محمدعلی مقیمی متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان‌نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

محمد علی مقیمی

امضا

تقدیم

به آنکه جز به فضلش امیدی نیست ...

سپاسگزاری

از پدر و مادرم که همواره در مواجهه با سختی‌های این دنیا دلسوزانه همراهم بوده‌اند؛

از استاد بزرگوارم جناب آقای دکتر حمیدرضا زرندی که با حسن خلق و گشاده‌رویی، رهنمودهای شبانه‌روزی خود را از من دریغ نکرده‌اند؛

و از سایر عزیزانی که در کنارشان این نتیجه حاصل آمد کمال تشکر و قدردانی را دارم.

چکیده

امروزه با توسعه زیرساخت‌های ابری لبه و اینترنت اشیا، و افزایش نیازهای برنامه‌ها به منابع سخت‌افزاری، برون‌سپاری وظایف بیش از پیش مورد توجه قرار گرفته است. رویکردهای متنوعی برای برون‌سپاری وظایف در گذشته پیشنهاد شده است که هر کدام به علت به برخی محدودیت‌هایی که داشتند، نتوانستند توسعه مطلوبی را تجربه کنند. در این پروژه، یک سیستم برون‌سپاری وظایف با رویکرد برون‌سپاری در سطح پردازه با تکیه بر ابزار CRIU توسعه داده شده است تا برخی از محدودیت‌های روش‌های پیشین را رفع کند. در این رویکرد پردازه‌هایی که احتیاج به برون‌سپاری دارند در کامپیوتر مبدأ متوقف شده، به کامپیوتر مقصد انتقال یافته و در آن اجرا می‌شوند، و پس از اتمام اجرا به کامپیوتر مبدأ بازمی‌گردند. این سیستم در ارزیابی با استفاده از برنامه‌های محک استاندارد، کارایی مناسبی را نشان داده است و در بهترین حالت، زمان اجرای کلی یکی از برنامه‌های محک با استفاده از این سیستم ۸۷ درصد کاهش داشته است. با برطرف کردن برخی محدودیت‌های سیستم برون‌سپاری، می‌توان توسعه مطلوبی را از آن انتظار داشت.

واژه‌های کلیدی:

برون‌سپاری وظایف- مهاجرت پردازه-نقطه وارسی-بازگردانی-رایانش ابری لبه

فهرست مطالب

فهرست شکل‌ها

صفحه

فهرست جدول‌ها

صفحه

فصل اول

مقدمه

مقدمه

محبوبیت و رشد استفاده از تلفن‌های همراه، دستگاه‌های اینترنت اشیاء، و کامپیوترهای شخصی در دهه اخیر که با توسعه فناوری اطلاعات همراه بوده، انتظارات کاربران را از وظایف محول شده به این دستگاه‌ها را تغییر داده است. امروزه انتظار می‌رود که کامپیوترهای شخصی و تلفن‌های همراه بتوانند وظایف متعددی از جمله تشخیص چهره، یادگیری ماشین، پردازش صوت، واقعیت افزوده و بازی‌های گرافیکی را انجام دهند [۱]. اجرای برخی از این وظایف، با توجه به منابع سخت‌افزاری محدود این دستگاه ها چالش‌هایی مانند زمان اجرای زیاد، و مصرف باتری زیادی و محدودیت طبیعی باتری این دستگاه ها ایجاد کرده است [۲].

توسعه روزافزون رایانش ابری، به‌ویژه رایانش ابری در لبه، هم‌زمان با بهبود کیفیت شبکه دسترسی ارائه شده به کاربران، باعث به وجود آمدن روش‌هایی برای حل این چالش‌ها شده است. یکی از این روش‌ها برون‌سپاری وظیفه[1] است. در این روش کامپیوترها، و دستگاه‌های اینترنت اشیاء که منابع سخت‌افزاری محدودی در اختیار دارند، می‌توانند بخشی از وظایف که نیاز به منابع سخت‌افزاری زیاد دارند را به ابر منتقل کرده تا در ابر که دارای منابع سخت‌افزاری قدرتمندتر است اجرا شود و بدین ترتیب، این وظایف در زمان کوتاه‌تری به اتمام برسند [۳].

برون‌سپاری وظیفه کاربردهای زیادی در حوزه‌های مختلف، از جمله رانندگی خودکار، واقعیت افزوده، سیستم‌های نظارتی و تقسیم بار پیدا کرده است. استفاده از این روش در بسیاری از کاربردها در مراحل ابتدایی توسعه است در نتیجه قابلیت توسعه بالایی دارد.

نحوه جداسازی بخش‌های برنامه که احتیاج به منابع سخت‌افزاری بیشتری دارند، از چالش‌های برون‌سپاری وظایف است. این کار اغلب در سطح بالا و توسط برنامه‌های کاربردی انجام می‌شود؛ به‌طوری که توابعی در ابر، توسط دستگاه‌های با منابع سخت‌افزاری محدود صدا زده می‌شود و ورودی های موردنیاز از این دستگاه‌های به ابر ارسال می‌شود، یا قطعه کد مورد نظر برای اجرا در ابر، توسط توسعه دهنده مشخص شده و به ابر فرستاده می‌شود. سپس در ابر این توابع اجرا می‌شود و پس از اتمام آن،

[1] task offloading

خروجی و نتایج این توابع به این دستگاه ها ارسال می‌شود [۱۱]. در این روش که به مهاجرت کد[1] موسوم است، مکانیزم برون‌سپاری وظیفه اغلب توسط برنامه کاربردی پیاده می‌شود. به‌علاوه در برخی موارد نیاز است این توابع در ابر از پیش تعریف شده باشند؛ لذا نیازمند وجود کد منبع[2] یا کد باینری وظیفه انتخاب شده برای ارجاع به ابر، در ابر می‌باشد [۱۱]. در نتیجه توسعه‌دهنده باید برای تعریف این توابع در ابر به طور مستقیم با ابر در تعامل باشد که در نهایت، باعث توسعه پذیری پایین این روش و محدودیت در استفاده عمومی از آن می‌شود.

برای حل این چالش، می‌توان ارسال این وظایف را از سطح برنامه‌های کاربردی به سطح سیستم‌عامل آورد؛ به‌طوری که جداسازی وظایف سنگین از سایر وظایف در سطح پردازه[3]ها در سیستم‌عامل انجام شود. این رویکرد دو مزیت اصلی می‌تواند به همراه داشته باشد:

۱. ازآنجاکه وظیفه‌ای که به ابر انتقال می‌یابد یک پردازه است؛ دیگری نیازی به تعامل مستقیم توسعه دهنده برنامه، با ابر به جهت فراهم آوردن امکانات و داده‌های اختصاصی موردنیاز برنامه، مانند کد منبع برنامه نیست

۲. با وجود اطلاعات آماری موجود از پردازه‌ها در حین اجرا در سیستم‌عامل، مانند میزان استفاده از پردازنده و حافظه می‌توان به طور خودکار پردازه‌هایی که مصرف بیشتری از منابع سخت‌افزاری دارند و برای برون‌سپاری مناسب هستند را تشخیص داد.

بااین‌وجود؛ همه پردازه‌های یک کامپیوتر قابلیت انتقال به کامپیوتر دیگر و اجرای صحیح در آن را ندارند. پردازه‌هایی که از حافظه مشترک با پردازه دیگر استفاده می‌کنند، پردازه‌هایی که از کاربر ورودی می‌گیرند و با آن تعامل می‌کنند، یا پردازه‌هایی که اجرای آنها در کامپیوتر دیگر، باعث عدم صحت عملکرد آن‌ها می‌شود (مانند آنتی‌ویروس‌ها)، از جمله پردازه‌هایی هستند که نمی‌توانند به کامپیوتر دیگری انتقال یابند؛ لذا یکی از چالش‌های اصلی این رویکرد، تشخیص قابل‌انتقال بودن پردازه‌های اعلامی توسط کاربر و انتخاب درست مجموعه پردازه‌ها برای انتقال به سرور است

[1] code migration

[2] source code

[3] process

در پژوهش های پیشین، رویکردهای مختلفی برای برون‌سپاری وظیفه بررسی شده است. همان‌طور که پیشتر اشاره شد؛ برون‌سپاری وظیفه اغلب با مکانیزم مهاجرت کد و در سطح برنامه‌های کاربردی انجام می‌شود [۱۲-۱۵]. این مکانیزم به دو دسته اصلی تقسیم می‌شود. در دسته اول، در کد منبع برنامه کاربردی، توابعی که در ابر موجود است صدا شده می‌شود و بعد از ارسال ورودی موردنیاز این توابع به ابر، خروجی این توابع را دریافت و در ادامه برنامه از آن استفاده می‌کند. این دسته که مکانیزم مشابهی با Remote Procedure Call دارد، دو چالش اصلی دارد:

۱. نیازمند وجود کد وظیفه ارجاع داده شده به ابر، در ابر می‌باشد [۱۱].

۲. در صورت عدم دسترسی به ابر، اجرای برنامه شکست می‌خورد [۱۱].

در دسته دوم، قطعه ای از کد که توسط توسعه دهنده برای ارسال به ابر مشخص می‌شود نیز به همراه ورودی های موردنیاز به ابر ارسال می‌شود و پس از اجرای آن، خروجی توسط میزبان دریافت می‌شود. این روش، وابستگی اجرای برنامه به ابر را کم می‌کند، اما سربار ارسال اطلاعات بیشتر می‌شود [۱۱]. با این حال چالش اصلی هر دو دسته ذکر شده، نیازمندی تغییر در کد این برنامه‌ها و تعامل مستقیم توسعه دهنده با ابر است [۱۱].

مکانیزم دیگری که برای برون‌سپاری وظیفه استفاده می‌شود بر مبنای انتقال یک متناظر[1] به ابر است. در این مکانیزم با استفاده از ابزارهای مجازی‌سازی، یک متناظر از میزبان، به ابر منتقل می‌شود و تمام وظایف در میزبان و ابر اجرا شده و نتیاج برخی وظایف که در ابر سریعتر به اتمام می‌رسند، به میزبان فرستاده شده تا میزبان از آن استفاده کند [۱۶, ۱۷]. بزرگترین چالش این روش، سربار داده بسیار زیاد آن برای انتقال تصویر[2] میزبان به ابر است [۱۱].

مکانیزم دیگری نیز برای برون‌سپاری وظایف برپایه انتقال ریسمان‌های ماشین مجازی پیشنهاد شده است. این نوع برون‌سپاری اغلب برای برون‌سپاری وظایف گوشی‌های همراه بر پایه سیستم‌عامل اندروید انجام شده است [۱۱] و نیازمند تغییرات بسیار زیاد در ماشین‌های مجازی فعلی اند [۱۱]. در برخی از پژوهش‌های مرتبط با این مکانیزم، با استفاده از مکانیزم حافظه مشترک توزیع شده، چالش‌های

[1] mirror

[2] image

مربوط به حافظه مشترک را مدیریت می‌کنند [۱۸]. چالش اصلی این مکانیزم، وابستگی بسیار زیاد به ماشین مجازی و کاهش کارایی ماشین مجازی بعد از تغییر یافتن است [۱۱].

در برخی پژوهش‌هایی که بستری برای برون‌سپاری وظیفه توسعه داده‌اند، سیاست‌های مختلف برای تصمیم‌گیری انتقال یک وظیفه به ابر را نیز بررسی کرده‌اند [۲, ۱۹]. برخی متغیر هایی که در تصمیم‌گیری موثر بوده اند شامل، پهنای باند بستر ارتباطی با سرور، تاخیر انتها به انتها تا سرور، توان پردازشی میزبان، و توان پردازشی سرور بوده است [۲, ۱۹].

از مکانیزم checkpoint/restore[1] برای تقسیم بار بین هسته‌های مختلف در سرورها استفاده می‌شود [۱۹]. ازآنجاکه بار روی هر هسته مطابق با نوسانات در منابع در دسترس و نیازهای پویای برنامه‌های در حال اجرا تغییر می‌کند؛ لازم است تا بار موجود روی آنها نیز به طور پویا تغییر کند تا تاخیر اجرای وظایف مختلف کم شده و از منابع به‌صورت کاراتری استفاده شود [۱۹]. برای انجام این کار، ابزارهای متفاوتی در سطح کاربر (مانند CRIU) و در سطح هسته[۲] (مانند OpenVZ) وجود دارد.

به‌علاوه، پژوهش‌هایی برای برون‌سپاری وظایف با مهاجرت پردازه برای دستگاه‌های اینترنت اشیاء انجام شده است تا انرژی مصرف شده در این دستگاه‌ها و زمان اجرای پردازه‌ها را کاهش دهند [۲]. به علت عدم وجود ماشین مجازی واحد در این دستگاه‌ها، برون‌سپاری وظایف در سطح سیستم‌عامل، وابسته به بستر اجرایی میزبان می‌شود [۲].

با این وجود، در بستر اینترنت اشیاء، دستگاه‌های انتهایی اغلب تنها برخی پردازش‌های ساده مانند تصفیه داده[۳]، وضعیت دهی سیگنال[۴]، تجمیع[۵]، حذف داده‌های پرت[۶] و تبدیل سیگنال آنالوگ به دیجیتال[۷] بر روی داده‌ها اتفاق می‌افتد. سایر تحلیل‌های پیچیده‌تر در ابر و همراه با سایر داده‌های انجام می‌پذیرد و

[1] نقطه وارسی/بازگردانی

[2] kernel

[3] filtering

[4] signal conditioning

[5] aggregation

[6] outlier dropout

[7] analog to digital convertion

دستورات کنترلی از ابر به دستگاه‌های انتهایی ارسال می‌شود. به عبارتی، در بستر اینترنت اشیاء، دستگاه‌های انتهایی معمولاً وظیفه تحلیل و تصمیم گیری بر اساس داده‌ها را ندارند، بلکه وظیفه جمع‌آوری داده‌ها، انجام پردازش‌های ساده بر روی آنها، انتقال داده‌ها به ابر، و انجام دستورات کنترلی ارسال شده از ابر را دارند [۲۰].

با توجه به توضیحات فوق، از آنجا که در بستر اینترنت اشیاء، دستگاه‌های انتهایی اغلب به نتایج تحلیل‌های مربوط به داده‌های خود احتیاج ندارند، بلکه به تصمیم‌های حاصل از این نتایج احتیاج دارند، استفاده از رویکرد مهاجرت پردازه برای این بستر چندان مناسب نیست. در این رویکرد، سعی می‌شود تا پردازه‌ها همان‌طور که در کارخواه هستند به کامپیوتر دیگر انتقال یابند تا پس از اتمام اجرا بتوانند به سادگی به کارخواه بازگردند. در نتیجه این رویکرد بیشتر مناسب استفاده در کامپیوترهای شخصی یا سایر مواردی که ایجاد کننده وظیفه، خود به طور مستقیم به نتایج آن وظیفه نیاز دارد می‌باشد.

تا کنون بستری به منظور برون‌سپاری وظیفه در سطح سیستم‌عامل با رویکرد مهاجرت پردازه و با استفاده از مکانیزم checkpoint/restore و انتقال پردازه‌ها از میزبان به سرور دارای توان پردازشی بیشتر مبتنی بر سیستم‌عامل لینوکس توسعه داده نشده است؛ لذا توسعه چنین بستری، که هدف اجرای این پروژه است، می‌تواند چالش مربوط به وجود کد باینری وظیفه در سرور برای انجام برون‌سپاری وظیفه را برای برنامه‌های کاربردی این سیستم‌عامل رفع کند.

در این پروژه، یک بستر بر پایه سیستم‌عامل لینوکس[1] برای برون‌سپاری وظایف در سطح پردازه توسعه داده شده است تا واحدهای اجرایی تعریف شده در سیستم‌عامل برای اجرا به ابر منتقل شوند. برای این کار پردازه‌هایی در سیستم‌عامل که ارسال آنها به ابر باعث کوتاه‌تر شدن زمان اتمام آن‌ها می‌شود، توسط توسعه‌دهنده در کد برنامه اعلام می‌شود. سپس، با استفاده از مکانیزم checkpoint/restore، از پردازه انتخاب شده checkpoint گرفته شده و اجرای آن در میزبان متوقف می‌شود. این checkpoint به همراه وابستگی‌های پردازه مانند فایل‌های مورداستفاده به ابر فرستاده شده و بازگردانی می‌شوند تا در ابر اجرا شوند. پس از اتمام این پردازه با اعلام خود پردازه مبنی بر بازگشت به میزبان، این پردازه به میزبان بازگردانده می‌شود تا نتایج آن در دسترس میزبان قرار گیرد.

[1] linux

در این پروژه به علت چالش‌های بیان شده، تشخیص و انتخاب پردازه‌ها برای برون‌سپاری به طور خودکار انجام نمی شـــود. در نتیجه تشـــخیص قابل انتقال بودن یک پردازه به کامپیوتر دیگر و انتخاب موقعیت مناسب برای انتقال پردازه به عهده توسعه دهنده است.

در فصل دوم به بررسی مشخصات سیستم برون‌سپاری وظیفه و اجزایی که این سیستم به آن‌ها نیاز دارد پرداخته می‌شود. در فصل سوم به تشریح پیاده‌سازی سیستم برون‌سپاری وظیفه و بررسی و تحلیل ابزارها مورد نیاز این سیستم پرداخته می‌شود و جزئیات پیاده‌سازی هر موجودیت و محدودیت‌های آن‌ها تشریح می‌شود. در فصل چهارم با استفاده از برنامه‌های مختلف، کارایی سیستم ارزیابی می‌شود. در فصل پنجم ویژگی‌ها و محدودیت‌های سیستم جمع‌بندی شده و پیشنهادهایی جهت بهبود عملکرد سیستم برای پژوهش‌های آینده ارائه می‌شود.

فصل دوم

طراحی و معماری سیستم برون‌سپاری

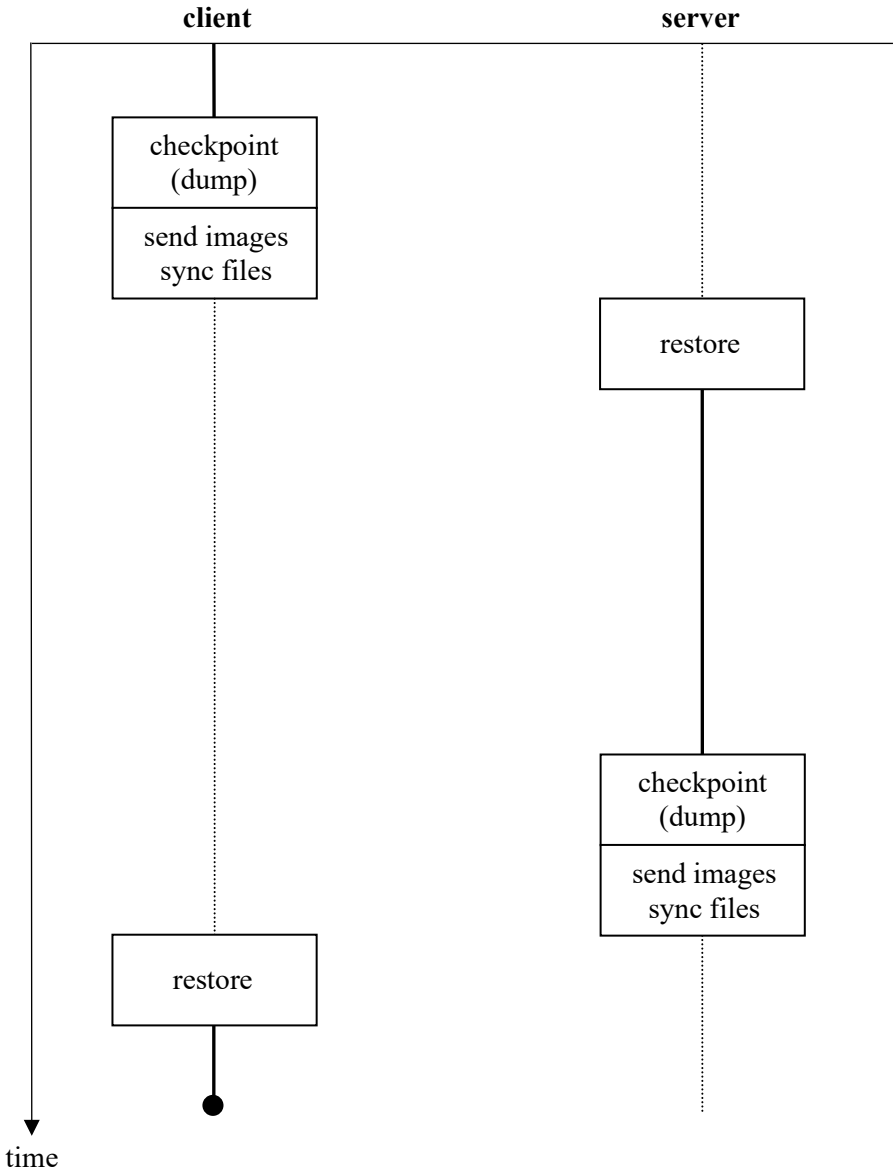
طراحی و معماری سیستم برون‌سپاری

در این فصل به بررسی مشخصات سیستم برون‌سپاری با توجه به عملیات‌های مورد نیاز برای انجام برون‌سپاری می‌پردازیم. در برون‌سپاری با رویکرد مهاجرت پردازه برای برون‌سپاری یک پردازه، در کامپیوتر کارخواه[1] از آن checkpoint گرفته و آن را به کامپیوتر کارساز منتقل می‌کنیم. سپس پس از اتمام اجرا در کامپیوتر کارساز[2]، آن پردازه را به‌صورت یک checkpoint به کامپیوتر کارخواه باز می‌گردانیم. به‌علاوه لازم است فایل‌هایی که پردازه در اجرا از آن استفاده می‌کند را نیز در محیط اجرا برای پردازه فراهم شود.

شکل۲-۱ نمودار جریان وقایع را در حین برون‌سپاری پردازه نشان می‌دهد. پردازه در کامپیوتر کارخواه شروع به اجرا می‌کند. سپس به تشخیص توسعه دهنده در نقطه‌ای از اجرای برنامه، پردازه درخواست برون‌سپاری را اعلام می‌کند. در این لحظه از پردازه checkpoint گرفته شده و اجرای پردازه در کامپیوتر کارخواه متوقف می‌شود. سپس فایل‌های تصویر مربوط به checkpoint پردازه و فایل‌هایی که پردازه برای اجرا به آنها متکی است به کامپیوتر کارساز منتقل می‌شود. سپس در کامپیوتر کارساز، پردازه restore شده و اجرای آن جریان می‌یابد. در انتهای اجرای پردازه نیز به تشخیص توسعه دهنده، پردازه درخواست بازگشت به کامپیوتر کارخواه را اعلام می‌کند. در این لحظه از پردازه checkpoint گرفته شده و اجرای پردازه در کامپیوتر کارساز متوقف می‌شود. سپس فایل‌های تصویر مربوط به checkpoint پردازه به کامپیوتر کارخواه منتقل شده و فایل‌هایی که پردازه برای اجرا به آنها متکی است نیز در صورت مغایرت در دو کامپیوتر، با هم هماهنگ می‌شود. سپس در کامپیوتر کارخواه، پردازه restore شده و تا اجرای پردازه به اتمام برسد.

[1] client

[2] server



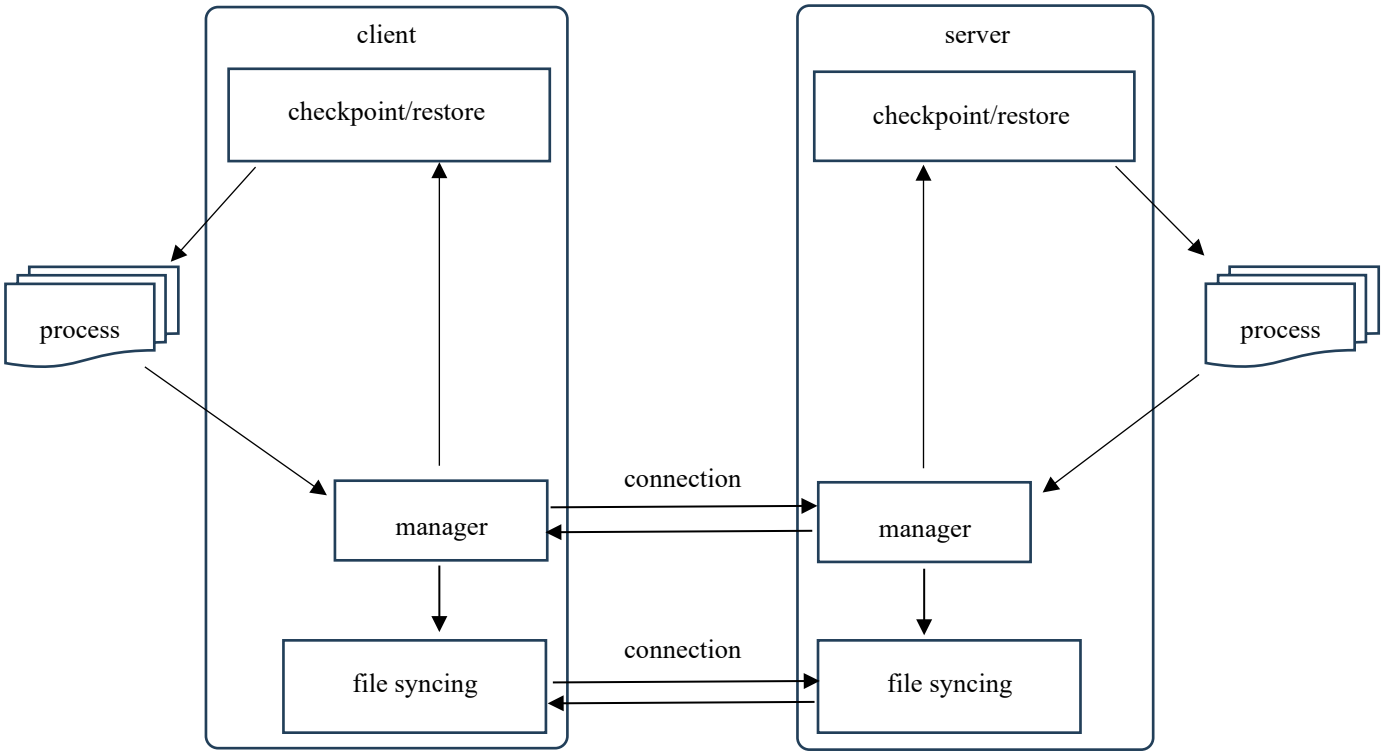
شکل۲-۱: جریان وقایع هنگام برون‌سپاری پردازه.

با توجه به شـــکل۲-۱، نیاز به موجودیت‌هایی داریم تا قادر به انجام عملیات‌های checkpoint، restore، و ارسال و هماهنگ‌سازی فایل‌ها باشند. به این منظور، سه موجودیت اصلی برای انجام وظایف مورد نیاز تعریف می‌شود:

- **موجودیت checkpoint/restore**: وظیفه انجام checkpoint و restore را به عهده دارد.

- **موجودیت هماهنگ‌سازی فایل‌ها**: وظیفه ارسال فایل‌های تصویر مربوط به checkpoint به کامپیوتر دیگر و هماهنگ‌سازی فایل‌های موردنیاز در دو کامپیوتر را انجام می‌دهد.
- **موجودیت مدیر**: وظیفه مدیریت و هماهنگی دو موجودیت قبلی، و هماهنگی با کامپیوتر دیگر جهت اجرای صحیح پردازه برون‌سپاری وظیفه را به عهده دارد.

شکل ۲-۲ معماری سیستم برون‌سپاری را نشان می‌دهد. همان‌طور که در این شکل نشان داده شده است، سه موجودیت با تعامل با پردازه‌ها برای انجام برون‌سپاری وظایف در هر کامپیوتر نقش ایفا می‌کنند. به‌علاوه، این سیستم دارای یک معماری متقارن در client و server است، به‌طوری که موجودیت‌ها در دو کامپیوتر یکسان هستند.. در ادامه این فصل به بررسی دقیق‌تر وظایف هر موجودیت و تعاملات لازم بین موجودیت‌ها و می‌پردازیم.



شکل۲-۲: معماری سیستم برون‌سپاری وظیفه.

۲-۱-موجودیت checkpoint/restore

برای انتقال یک پردازه به کامپیوتر دیگر، ابتدا لازم اســـت این موجودیت اجرای پردازه متوقف کرده و اطلاعات لازم برای بازگردانی پردازه را ذخیره کند (checkpoint). سپس لازم است این اطلاعات به کامپیوتر دیگر منتقل شـــود تا این موجودیت در کامپیوتر دیگر با اســـتفاده از این اطلاعات، پردازه را بازگردانی کند (restore).

موجودیت checkpoint/restore در زمان مناســـب دســـتور انجام عملیات checkpoint یا restore یک پردازه را از موجودیت مدیر دریافت می‌کند. ســـپس با تعامل با پردازه مورد نظر، عملیات مناســـب را روی پردازه انجام می‌دهد. در نهایت موفقیت یا عدم موفقیت انجام عملیات را به مدیر اطلاع می‌دهد.

۲-۲- موجودیت هماهنگ‌سازی فایل‌ها

این موجودیت وظیفه ارسال و هماهنگ‌سازی دو دسته فایل را بین دو کامپیوتر را به عهده دارد:

۱. فایل‌هایی که فرایند برای اجرا به آن‌ها متکی است
۲. فایل‌های حاصل از عملیات checkpoint بر روی پردازه

به این منظور، این موجودیت در زمان مناسب دستور هماهنگ‌سازی فایل‌ها را از موجودیت مدیر دریافت می‌کند. در این زمان موجودیت مدیر، آدرس فایل‌هایی که نیاز به هماهنگ‌ســـازی دارند به علاوه آدرس کامپیوتر مقصـــد را به این موجودیت ارســـال می‌کند. ســـپس این موجودیت بطور مســـتقیم به موجودیت هماهنگ‌سازی فایل کامپیوتر متناظر ارتباط برقرار می‌کند تا عملیات هماهنگ‌سازی فایل‌های مورد نظر را انجام دهد.

۲-۳- موجودیت مدیر

وظیفه این موجودیت هماهنگی بین سایر اجزا سیستم برون‌سپاری به منظور اجرا صحیح فرایند برون‌سپاری است. این موجودیت در زمان مناسب، دستورات لازم را به سایر اجزا ارسال می‌کند و موفقیت یا عدم موفقیت سایر اجزا در اجرای دستورات را دریافت می‌کند.

همانطور که در شکل۲-۲ نشان داده شده است، این موجودیت با چهار موجودیت دیگر در تعامل است. موجودیت مدیر با موجودیت‌های هماهنگ‌سازی فایل و checkpoint/rstore در تعامل است تا در زمان مناسب، دستورات لازم را برای انجام فعالیت‌های این دو موجودیت ارسال کند. به علاوه این موجودیت با پردازه‌ها در ارتباط است تا پردازه‌ها بتوانند در زمان مناسب درخواست برون‌سپاری خود را به این موجودیت اعلام کنند. همچنین موجودیت مدیر، با موجودیت مدیر کامپیوتر متناظر نیز در ارتباط است. این ارتباط به این منظور است تا پس از آن که اجرای پردازه در کامپیوتر مبدأ متوقف شد و داده‌های حاصل از عملیات checkpoint به کامپیوتر مقصد منتقل شد، مدیر کامپیوتر مبدأ به مدیر کامپیوتر مقصد اطلاع دهد که تمام داده‌های مورد نیاز برای بازگردانی پردازه مورد نظر به کامپیوتر مقصد منتقل شده است و این پردازه آماده بازگردانی در کامپیوتر مقصد است.

جریان اجرا در موجودیت مدیر به اینصورت است که این موجودیت ابتدا درخواست برون‌سپاری از یک پردازه دریافت می‌کند. سپس با هماهنگی و ارسال دستورات مناسب به موجودیت‌های checkpoint/restore و هماهنگ‌سازی فایل، از پردازه checkpoint گرفته شده و داده‌ها و فایل‌های مورد نیاز، به کامپیوتر مقصد منتقل می‌شوند. در نهایت مدیر کامپیوتر مبدأ یک اعلان به منظور آمادگی پردازه مورد نظر برای بازگردانی در کامپیوتر مقصد به مدیر کامپیوتر مقصد ارسال می‌کند. مدیر کامپیوتر مقصد پس از دریافت این اعلان، با ارسال دستورات لازم به موجودیت checkpoint/restore، پردازه مورد نظر را بازگردانی می‌کند. در نتیجه لازم است موجودیت مدیر همواره آماده دریافت اعلان از پردازه‌ها و از مدیر کامپیوتر متناظر باشد. شایان ذکر است که اعلان دریافتی از پردازه‌ها، یک اعلان محلی است که از یک پردازه در کامپیوتر به پردازه دیگر ارسال می‌شود؛ اما اعلان دریافتی از موجودیت مدیر کامپیوتر متناظر محلی نیست و لازم است در بستر شبکه به کامپیوتر متناظر ارسال شود.

در فصل آینده به بررسی پیاده‌سازی سیستم برون‌سپاری پرداخته می‌شود. به این منظور ابتدا ابزارهای از پیش توسعه داده شده که مناسب اجرای وظایف مورد نیاز سیستم برون‌سپاری هستند بررسی و تحلیل می‌شوند. سپس درباره پیاده سازی هر یک از موجودیت‌ها توضیحات مربوطه ارائه می‌شود.

فصل سوم

پیاده‌سازی سیستم برون‌سپاری

پیاده‌سازی سیستم برون‌سپاری

در فصل گذشته عملیات‌های لازم برای برون‌سپاری در سطح پردازه بررسی شد و ویژگی‌ها و موجودیت‌های مورد نیاز برای پیاده‌سازی این سیستم معرفی شد. در این فصل به تشریح روش پیاده‌سازی سیستم برون‌سپاری پرداخته می‌شود.

با توجه به شکل۲-۲، پیاده سازی سه موجودیت برای توسعه سیستم برون‌سپاری ضروری است. به این منظور یا می‌توانیم ابزارهای موجود که قادر به انجام این فعالیت‌ها هستند را بررسی و در صورت دارا بودن توانایی‌های مورد نیاز سیستم برون‌سپاری، از آنها استفاده کنیم، و یا خود از ابتدا ابزارهایی را به این منظور توسعه دهیم. در این پروژه تا حد امکان از ابزارهای آماده برای انجام عملیات های مورد نیاز استفاده شد. استفاده از ابزارهای آماده علاوه بر صرفه‌جویی در زمان پیاده‌سازی، به علت اینکه توسط کاربران متعددی استفاده می‌شوند اغلب قابلیت اطمینان بیشتری دارد و باعث افزایش قابلیت توسعه این سیستم در آینده می‌شود.

۳-۱-موجودیت checkpoint/restore

برای انتقال یک پردازه از یک کامپیوتر به کامپیوتر دیگر، لازم است تا تمام اطلاعات مربوط به پردازه و وضعیت آن مانند فضای حافظه، پرچم‌ها[1] و فایل‌های مورد استفاده ذخیره شده و به کامپیوتر کارساز منتقل شود. سپس با استفاده از این اطلاعات در کامپیوتر کارساز، پردازه بازیابی شود. برای انجام این دو عملیات، ابزارهای مختلفی توسعه داده شده‌اند. برخی از ابزارها که در این پروژه مورد بررسی قرار گرفته‌اند عبارت اند از:

- **ابزار DMTCP**[2]: این ابزار یک کتابخانه برای گرفتن checkpoint و restore پردازه‌ها در لینوکس است. این ابزار برای برنامه‌هایی که با کتابخانه‌های DMTCP سازگار هستند، مانند MPI، OpenMP، MATLAB، Python، Perl، R، و بسیاری دیگر از زبان‌های

[1] flags

[2] Distributed MultiThreaded CheckPointing

برنامه‌نویسی مناسب است، اما ممکن است مشکلات عملکردی و پیچیدگی‌هایی در بازگردانی پردازه‌ها داشته باشد. برای استفاده از این ابزار، کتابخانه مربوط به آن باید در کد پردازه‌ها بارگذاری شده باشد [۲۱, ۲۲].

- **ابزار BLCR**[1]: این ابزار، یک ابزار سطح هسته سیستم‌عامل برای گرفتن checkpoint و restore پردازه‌ها است و به‌صورت یک ماژول هسته و برای هسته‌های لینوکس پیاده‌سازی شده است و از معماری‌های مختلفی مانند x86، x86_64، PPC/PPC64 و ARM پشتیبانی می‌کند [۲۳]. این ابزار به طور گسترده استفاده نمی شود و از سال ۲۰۱۳ میلادی بروزرسانی دریافت نکرده است.
- **ابزار PinPlay**: این ابزار وضعیت رجیسترهای پردازنده و تمام صفحات حافظه‌ای که شامل کد برنامه و کتابخانه‌های مشترک هستند را ذخیره می‌کند. در برخی موارد که عملیات‌های checkpoint و restore روی پردازه با فضای حافظه زیادی اجرا می‌شود، اجرای این عملیات‌ها با سربار زمانی بسیار زیاد یا با خطا رو به رو می‌شود [۲۱].
- **ابزار OpenVZ**: این ابزار، یک ابزار مجازی‌سازی سطح هسته برای لینوکس است که قابلیت گرفتن checkpoint و restore پردازه‌ها را دارد. این ابزار به‌صورت یک ماژول هسته پیاده‌سازی شده است و از ویژگی‌های مختلف هسته برای مدیریت منابع پردازه‌ها استفاده می‌کند. از مزایا این ابزار به سرعت زیاد هنگام انجام این عملیات‌ها می‌توان اشاره کرد [۲۴].
- **ابزار CRIU**[2]: این ابزار، یک ابزار توسعه داده شده برای لینوکس است که می‌تواند یک برنامه را متوقف کرده و وضعیت آن را در غالب فایل‌های تصویر که شامل اطلاعاتی مانند حافظه، رجیسترها، فایل‌های باز، اتصالات شبکه و سایر منابع مورد استفاده پردازه هستند در حافظه ذخیره کند. سپس داده‌های ذخیره شده می‌توانند برای بازگردانی برنامه و اجرای آن دقیقاً همان‌طور که در زمان توقف بوده است، استفاده شود [۲۱]. برخی مزایا CRIU نسبت به سایر ابزارهای مشابه آن عبارت‌اند از [۲۱]:

[1] Berkeley Lab Checkpoint/Restart

[2] Checkpoint Restore In Userspace

- **کارایی در شـــرایط و برنامه‌های مختلف:** CRIU نیازی به اتصـــال به کتابخانه‌های خاص یا تغییرات در کد برنامه‌ها ندارد. این ابزار می‌تواند هر برنامه‌ای را که توسط هسته پشتیبانی می‌شود، checkpoint و restore کند.
- **پشـتیبانی از ویژگی‌های پیشـرفته هسـته:** CRIU از ویژگی‌های پیشـرفته هسـته لینوکس که به تازگی اضـــافه شـــده‌اند، مانند namespaceها و cgroupها پشـــتیبانی می‌کند.
- **پشـتیبانی از مخزن‌ها**[1]: CRIU می‌تواند پردازه‌های درون مخزن‌ها را نیز checkpoint و restore کند، که این ویژگی برای محیط‌های مجازی‌ســـازی و کانتینری بســـیار مفید است.
- **مدیریت منابع پیشـرفته:** CRIU می‌تواند منابع مختلفی مانند فایل‌های باز، اتصـالات شـــبکه، تایمرها و ســـیگنال‌ها را به درســـتی مدیریت کند و آن‌ها را در زمان restore بازگرداند.
- **اسـتفاده گسـترده:** این ابزار، پراسـتفاده‌ترین ابزار برای انجام عملیات‌های checkpoint و restore در سـطح کاربر اسـت و جامعه اسـتفاده کنندگان بزرگ‌تری نسـبت به سـایر ابزارها دارد. در نتیجه در صـــورت برخورد با خطا یا چالش در حین اســـتفاده، راحت‌تر می‌توان آن‌ها را رفع کرد.

با توجه به موارد بیان شـده، در این پروژه از ابزار CRIU برای انجام عملیات‌های checkpoint و restore اسـتفاده شـد. برای پیاده‌سـازی این سـیسـتم از دو عملگر checkpoint و restore این ابزار استفاده شده است که این دو عملگر به طور دقیق‌تر بررسی می‌شوند.

[1] container

۳-۱-۱-عملگر checkpoint

عملگر checkpoint، وابستگی زیادی به فایل‌های موجود در آدرس proc/ سیستم‌عامل لینوکس دارند. به طور کلی، CRIU عمده اطلاعات موردنیاز خود را از یک پردازه، مانند اشاره‌گر فایل‌ها[1]، لوله‌ها[2]، و نگاشت‌های حافظه[3] را از فایل‌های موجود در این آدرس می‌خواند [۳۰].

درCRIU با دستور dump، از یک پردازه checkpoint گرفته می‌شود. برای انجام این عملیات، ابتدا با پیمایش در آدرس /proc/$pid/task/$tid/children تمام فرزندان و سپس خود پردازه با دستور PTRACE_SEIZE متوقف می‌شوند. سپس تمام اطلاعات موردنیاز خوانده شده و در فایل‌های تصویر ذخیره می‌شوند [۳۰]. در نهایت می‌توان با استفاده از ptrace، پردازه متوقف شده را به جریان انداخت یا با استفاده از SIGKILL، پردازه را در سیستم‌عامل از بین برد. تابع dump به طور پیشفرض پردازه را از بین می برد [۲۱].

در این پروژه، فایل‌های تصویر مربوط به پردازه، در آدرس /home/plto/images_dir/$pid/ ذخیره می‌شوند تا در زمان مناسب، کل پوشه $pid به کامپیوتر کارساز منتقل شده تا پردازه بازیابی شود.

۳-۱-۲-عملگر restore

در restore، CRIU پردازه اجرایی توسط خود را طی مراحلی تبدیل به پردازه مورد نظر می‌کند. برای انجام این کار، CRIU ابتدا با استفاده از فایل‌های تصویر، منابع مشترک بین پردازه‌های checkpoint شده را ایجاد می‌کند. سپس با استفاده از تابع fork، مجموعه پردازه‌های checkpoint شده ایجاد می‌شود. سپس تمامی منابع مانند اشاره‌گرهای فایل، نگاشت‌های حافظه، زمان‌سنج‌ها[4] بازیابی

[1] file pointers

[2] pipes

[3] memory maps

[4] timers

می‌شوند. پردازه بازیابی شده شناسه پردازه[1] یکسان با همان پردازه قبل از checkpoint و در کامپیوتر کارخواه را دارد [۲۱, ۳۰].

در این پروژه پس از انتقال فایل‌های تصویر برای انجام عملیات restore، یک پردازه جدید ایجاد شده و در آن GNOME Terminal اجرا می‌شود. سپس دستور restore در آن ترمینال اجرا می‌شود تا پردازه در این محیط بازیابی شود. استفاده از GNOME Terminal از این جهت ضروری است که در کامپیوتر کارخواه، برای اجرای پردازه‌ها از آن استفاده می‌شود. به عبارتی، چون پردازه‌ها در کامپیوتر کارخواه در GNOME Terminal شروع به اجرا می‌کنند، این ترمینال پردازه والد آن برنامه‌ها محسوب می‌شود و در نتیجه برای restore کردن پردازه، لازم است آن نیز به عنوان یکی از منابع در اختیار پردازه قرار گیرد.

۳-۱-۳-محدودیت‌ها

در برخی شرایط، این ابزار نمی‌تواند به طور صحیح از یک پردازه checkpoint گرفته و سپس آن را restore کند. برخی مواردی که موجب این محدودیت می‌شوند عبارت‌اند از [۲۱]:

- **شناسه پردازه در کارساز موجود باشد:** چون پردازه با همان شناسه پردازه که در کارخواه داشت، در کارساز بازیابی می‌شود، در صورتی که این شناسه پردازه در کامپیوتر کارساز به یک پردازه در حال اجرا تعلق داشته باشد، امکان بازیابی پردازه وجود ندارد.
- **پردازه دارای سوکت باز باشد:** در این شرایط، به جهت این که این سوکت هنگام توقف پردازه از بین می‌رود، عملکرد صحیح برنامه نمی‌تواند تضمین شود.
- **پردازه به اشکال‌یاب[2] متصل باشد:** چون CRIU و اشکال‌یاب‌ها هر دو از APIهای یکسانی برای اتصال به پردازه و گرفتن وضعیت آن استفاده می‌کنند و این API اجازه استفاده هم‌زمان از آن را به بیش از یک پردازه نمی‌دهد، checkpoint گرفتن در این شرایط با خطا همراه می‌شود.

[1] PID

[2] debugger

- **ساختار فایل‌ها در هنگام checkpoint و restore با هم تفاوت داشته باشند:** در صورتی که ساختار یا محتوای فایل‌هایی که پردازه در لحظه checkpoint به آنها متکی است در هنگام restore متفاوت باشد، restore کردن پردازه با خطا همراه می‌شود.

در نتیجه در این پروژه در شرایط فوق، امکان برون‌سپاری وظیفه وجود ندارد. با این وجود، ازآنجاکه عدم هماهنگی فایل‌ها در کامپیوتر کارخواه و کارساز رایج است، برای غلبه بر این مشکل از ابزاری برای هماهنگ‌سازی فایل‌هایی که پردازه به آنها متکی است در کامپیوتر کارخواه و کارساز استفاده می‌کنیم.

۳-۲-موجودیت هماهنگ‌سازی فایل‌ها

با توجه به شکل۲-۱، برای انتقال فایل‌های تصویر و سایر فایل‌هایی که پردازه برای اجرا به آنها متکی است، لازم به استفاده از ابزاری هستیم تا هماهنگ‌سازی این فایل‌ها را بین دو کامپیوتر انجام دهد. برخی از این فایل‌ها در هر دو کامپیوتر پیش از انجام برون‌سپاری وجود دارند و نیازی به انتقال آنها نیست. برخی فایل‌ها نیز، تفاوت‌های جزئی با فایل متناظر آن‌ها در کامپیوتر دیگر دارند، لذا نیازی به انتقال تمام بخش‌های این فایل‌ها نیز وجود ندارد. با توجه به شرایط بیان شده، در این پروژه استفاده از ابزاری که زمان و سربار شبکه کمتری حین هماهنگ‌سازی فایل‌ها را دارد مطلوب است. به منظور انتخاب ابزار مناسب با ویژگی‌های گفته شده، برخی از این ابزارها بررسی و با یکدیگر مقایسه شده‌اند:

- **ابزار SCP**[1]: این ابزار، یک ابزار کاربردی در سیستم‌عامل لینوکس است که برای انتقال فایل بین دو سیستم استفاده می‌شود. این ابزار از پروتکل SFTP بر روی پروتکل SSH برای انتقال اطلاعات استفاده می‌کند. این ابزار یک فایل را به طور کامل از کارخواه به کارساز ارسال می‌کند و شباهت دو فایل در کارخواه و کارساز را در نظر نمی‌گیرد [۲۵].
- **ابزار Syncthing**: این ابزار، یک برنامه متن‌باز برای هماهنگ‌سازی فایل بین دو یا چند کامپیوتر است. این ابزار از پروتکل BEP[2] برای هماهنگ‌سازی فایل‌ها استفاده می‌کند. در

[1] Secure Copy Protocol

[2] Block Exchange Protocol

این پروتکل داده‌ها به بخش‌هایی تقسیم می‌شوند. سپس هر کامپیوتر بخش‌هایی را که ندارد یا نسخه متفاوتی از آن را دارد، درخواست می‌کند [۲۶].

- **ابزار FreeFileSync**: این ابزار، یک ابزار مقایسه و هماهنگ‌سازی پوشه است به جای کپی کردن هر فایل در هر زمان، FreeFileSync تفاوت‌ها بین یک پوشه منبع و یک پوشه هدف را تعیین می‌کند و حداقل داده لازم را برای هماهنگ‌سازی منتقل می‌کند [۲۷].
- **ابزار Rclone**: این ابزار، یک برنامه خط فرمان برای هماهنگ‌سازی فایل‌ها است که بیشتر برای مدیریت فایل‌ها در ذخیره سازی ابر بکار می‌رود. این ابزار از ویژگی‌هایی مانند فشرده سازی و رمزنگاری پشتیبانی می‌کند. این ابزار عملکردی مشابه ابزار rsync دارد و تنها تفاوت فایل‌ها در دو کامپیوتر را ارسال می‌کند [۲۸].
- **ابزار rsync**: این ابزار، یک ابزار هماهنگ‌سازی فایل برای کپی فایل‌ها بین سیستم‌های محلی و راه دور است. این ابزار با استفاده از یک الگوریتم سریع، مقدار داده کپی شده را به حداقل می‌رساند و فقط بخش‌هایی از فایل‌ها که تغییر کرده‌اند را منتقل می‌کند. این ابزار از پروتکل SSH برای برقراری ارتباط با کامپیوترهای دیگر استفاده می‌کند. به‌علاوه این ابزار دارای ویژگی‌های متنوعی، مانند فشرده سازی داده‌های ارسالی می‌باشد [۲۹].

با توجه به موارد بیان شده، عملکرد برخی از این ابزارها شباهت زیادی با یکدیگر دارند. با این وجود در این پروژه به علت نصب ابزار rsync در اغلب توزیع‌های لینوکس به طور پیش فرض، سادگی استفاده از این ابزار، و ارائه ویژگی‌هایی مانند فشرده سازی داده ارسالی، از این ابزار جهت هماهنگ‌سازی فایل‌ها استفاده شده است.

همان‌طور پیش‌تر بیان شد، برای هماهنگ‌سازی فایل‌ها در دو کامپیوتر مجزا، این ابزار از ارتباط SSH استفاده می‌کند؛ در نتیجه لازم است که سرور SSH در دو کامپیوتر در حال اجرا باشد. rsync ویژگی‌های مختلفی را برای انتقال و هماهنگ‌سازی فایل‌ها می‌تولند در نظر بگیرد و از الگوریتمی برای هماهنگ‌سازی فایل‌ها استفاده می‌کند که سرعت این کار را افزایش می‌دهد [۲۹].

۳-۲-۱-الگوریتم سریع rsync

فرض کنید کامپیوتر a می‌خواهد فایل f_aرا به کامپیوتر b بفرستد. کامپیوتر b نیز دارای فایل f_b است که این فایل، مشابه همان فایل f_aیا تغییرات جزئی است. برای این کار، این الگوریتم مراحل زیر را انجام می‌دهد [۳۱]:

۱. فایل f_b به مجموعه‌ای از قسمت‌های غیرهم‌پوشان با اندازه ثابت S بایت تقسیم می‌شود. قسمت آخر ممکن است کوتاه‌تر از S بایت باشد.
۲. برای هر یک از این قسمت‌ها، یک جمع کنترلی[1] ضعیف ۳۲ بیتی و یک جمع کنترلی قوی ۱۲۸ بیتی محاسبه می‌شود.
۳. b این جمع کنترلی‌ها را به a ارسال می‌کند.
۴. a در f_a جستجو می‌کند تا تمام قسمت‌های با طول S بایت (در هر جابه‌جایی، نه فقط مضرب‌های S) که همان جمع کنترلی ضعیف و قوی یکی از قسمت‌های f_bرا دارند، پیدا کند.
۵. a یک دنباله از دستورالعمل‌ها برای ساخت یک رونوشت از f_a به b ارسال می‌کند. هر دستورالعمل یا یک اشاره‌گر به یک قسمت از f_b است، یا داده‌های واقعی از f_a داده‌های واقعی فقط برای قسمت‌هایی از f_aکه با هیچ یک از قسمت‌های f_bمطابقت نداشتند، ارسال می‌شود.

بدین ترتیب حجم داده بسیار کمی در مواردی که دو فایل در کارخواه و کارساز شباهت زیادی دارند مبادله می‌شود. این ویژگی برای هماهنگ‌سازی فایل‌هایی که پردازه به آنها متکی است، و برای انتقال فایل‌های تصویر از کامپیوتر کارساز به کامپیوتر کارخواه هنگام بازگرداندن پردازه بسیار مطلوب است؛ زیرا در این شرایط، فایل‌ها در کارخواه و کارساز شباهت زیادی با یکدیگر دارند.

در این پروژه از نسخه rsync ۳.۲ استفاده شده که قابلیت ساخت مسیر در کارساز در صورت عدم وجود را دارد. همچنین از ویژگی فشرده‌سازی این ابزار برای ارسال تمام داده‌ها استفاده شده است. ازآنجاکه rsync بر روی SSH اجرا می‌شود، آدرس IP و رمز عبور هر کامپیوتر برای کامپیوتری که با آن

[1] checksum

ارتباط برقرار می‌کند باید فراهم باشد. در این پروژه برای وارد کردن رمز عبور از sshpass استفاده شده است.

۳-۳-موجودیت مدیر

لازم است تا ابزارهای هماهنگ‌سازی فایل‌ها به‌خوبی مدیریت شوند تا عملیات‌های لازم برای برون‌سپاری در زمان مناسب اجرا شوند؛ بنابراین لازم است تا برنامه‌ای جهت مدیریت ابزارها توسعه یابد. برای توسعه این برنامه، به دلیل برخی ویژگی‌های زبان Go، از این زبان استفاده شد. با این وجود از سایر زبان‌های برنامه‌نویسی نیز برای توسعه این برنامه می‌توان استفاده کرد. برخی ویژگی‌های زبان Go که در انتخاب این زبان برای توسعه موجودیت مدیر موثر بودند عبارت‌اند از:

- **سادگی و خوانایی:** زبان Go با تمرکز بر سادگی و خوانایی کد، به توسعه دهندگان اجازه می‌دهد که به راحتی کدی را ایجاد کنند که قابل فهم و توسعه آن ساده و سریع باشد.
- **کارایی بالا:** زبان Go دارای سرعت اجرای بالا است و بهینه سازی خوبی برای استفاده از حافظه و پردازنده دارد.
- **وجود کتابخلنه‌های متنوع:** Go دارای کتابخانه‌های استلندارد قدرتمندی است که پشتیبانی قوی از شبکه، هم‌روندی[1]، رمزنگاری، پردازش متن و بسیاری از ویژگی‌های دیگر را فراهم می‌کند.
- **پشتیبانی از هم‌روندی:** با استفاده از زبان Go، به‌سادگی می‌توان برنامه‌های هم‌روند، با کارایی مناسب ایجاد کرد.
- **عدم نیاز به مفسر[2]:** ازآنجاکه این برنامه قبل از اجرا به کد ماشین تبدیل می‌شود، در زمان اجرا نیاز به مفسر ندارد. این ویژگی در کامپیوترهایی که منابع بسیار محدودی دارند و به سختی می‌توانند مفسرها را اجرا کنند دارای اهمیت بیشتری است.

[1] concurency

[2] interpreter

همانطور که پیش‌تر بیان شد، مدیر وظیفه هماهنگی بین پردازه، ابزار هماهنگ‌سازی فایل‌ها، و ابزار checkpoint/restore را به عهده دارد. به‌علاوه این موجودیت با ارتباط موجودیت مدیر کامپیوتر متناظر، آن را از ارسال یک پردازه برای برون‌سپاری با خبر می‌سازد. مدیر هم‌زمان دو سرور TCP برای ارتباط با پردازه‌های محلی در حال اجرا و مدیر کامپیوتر مقابل اجرا می‌کند.

۳-۳-۱-سرور TCP برای ارتباط با پردازه‌های محلی

این سرور بر روی درگاه[1] ۹۱۹۱ (بر روی درگاه‌های دیگر نیز می‌تواند اجرا شود، اما این درگاه باید با پردازه هماهنگ شود) آدرس IP محلی کامپیوتر اجرا می‌شود (بدین ترتیب فقط توسط پردازه‌های داخل این کامپیوتر در دسترس است).

پردازه‌ها برای ارتباط با مدیر باید یک سوکت درست کرده و از طریق آن داده‌های خود را به مدیر بفرستند. پس ارسال دستور مناسب توسط پردازه به مدیر، این سوکت بسته شده تا پردازه بتواند بدون خطا به کامپیوتر دیگر انتقال یابد. مدیر داده‌ای به پردازه‌ها ارسال نمی‌کند درنتیجه ارتباط پردازه‌ها با مدیر یک‌طرفه است.

شکل۳-۱ شبه کد این بخش از موجودیت مدیر را نشان می‌دهد. هر پردازه می‌تواند در زمان مناسب برای برون‌سپاری دستور "STP" را به همراه شناسه پردازه خود به مدیر ارسال کند. مدیر پس از دریافت این دستور، ابتدا تمامی فایل‌هایی که پردازه در این لحظه به آنها متکی است را با دستور lsof دریافت می‌کند. سپس فرمان checkpoint گرفتن از این پردازه را به ابزار checkpoint/restore می‌دهد. پس از اتمام گرفتن checkpoint، مدیر لیستی از فایل‌هایی که پردازه به آنها متکی است را به همراه فایل‌های تصویر حاصل از checkpoint این پردازه را به ابزار هماهنگ‌سازی فایل‌ها می‌دهد تا این ابزار این فایل‌ها را به کامپیوتر کارساز منتقل کند. در نهایت پس از انتقال کامل تمامی فایل‌ها به کامپیوتر کارساز، مدیر با اتصال به سرور TCP کامپیوتر کارساز که برای ارتباط با مدیر کامپیوترهای

[1] port

دیگر است، شناسه پردازه را به آن می‌فرستد. کامپیوتر کارساز با دریافت این شناسه پردازه، می‌تواند restore این پردازه را شروع کند.

```
1  handleSend(c connection):
2      command, pid ← read()
3      if command = "STP", then
4          depenantFiles ← getDepenamdFiles(pid)
5          imagesDirectory ← checkpoint(pid)
6          syncFiles(depenantFiles, imagesDirectory, destinationIP)
7          sendSignal(pid, destinationIP)
8      end
9  end
```

شکل۳-۱: شبه کد سرور TCP برای ارتباط با پردازه‌های محلی.

۳-۳-۲-سرور TCP برای ارتباط با مدیر کامپیوترهای دیگر

این سرور بر روی درگاه ۹۲۹۲ آدرس IP عمومی کامپیوتر اجرا می‌شود تا توسط سایر کامپیوترها قابل دسترس باشد. شکل ۳-۲ شبه کد مربوط به این بخش از موجودیت مدیر را نشان می‌دهد. دریافت یک شناسه پردازه توسط این سرور، به این معناست که تمامی فایل‌های موردنیاز و فایل‌های تصویر مربوط به این پردازه توسط کامپیوتر کارساز دریافت شده است و می‌توان در این کامپیوتر این پردازه را restore کرد. در نتیجه پس از دریافت شناسه پردازه، یک پردازه جدید در کامپیوتر کارساز ایجاد می‌شود، در آن پردازه GNOME Terminal اجرا شده و در این ترمینال به ابزار checkpoint/restore، فرمان restore کردن این پردازه فرستاده شود تا ابزار checkpoint/restore، پردازه‌ی که جهت برون‌سپاری به این کامپیوتر انتقال یافته است را بر روی این پردازه جدید restore کند.

```
1  handleRecieve(c connection):
2      pid ← read()
3      fork()
4      if it is the child process, then
5          restore(pid)
6      end
7  end
```

شکل۳-۲: شبه کد سرور TCP برای ارتباط با مدیر کامپیوترهای دیگر.

لازم به ذکر است موجودیت مدیر، بدون حالت[1] است؛ در نتیجه در صورت بروز خطا در هر بخش، ادامه پردازه آن متوقف می‌شود و مکانیزمی جهت بازیابی از خطا و رسیدن به حالت ایمن ندارد. این نوع توسعه در راستای ساده شدن پیاده‌سازی و عیب یابی سیستم برون‌سپاری گرفته شده است. برای توسعه این سیستم به جهت اجرا در محیط عملیاتی، می‌توان این نوع توسعه را تغییر داد تا سیستم برون‌سپاری قابلیت اطمینان بیشتری نسبت در هنگام بروز خطا داشته باشد.

۳-۳-۳-ملاحظات هنگام برون‌سپاری پردازه

همان‌طور که پیش‌تر بیان شد، درخواست برون‌سپاری پردازه توسط خود پردازه فرستاده می‌شود؛ درنتیجه توسعه دهنده برنامه وظیفه تشخیص مناسب بودن یک پردازه برای برون‌سپاری و سازگاری پردازه، با محدودیت‌های سیستم برون‌سپاری که در بخش۳-۱-۳ اشاره شده است را به عهده دارد. به‌علاوه، لازم است دستور ارسال این درخواست توسط توسعه دهنده در موقعیت مناسب کد منبع برنامه تعبیه شده باشد. توجه به موارد زیر هنگام تعبیه این درخواست در کد منبع حائز اهمیت است:

- در هر برنامه برای برون‌سپاری پردازه مربوط به آن، درخواست برون‌سپاری دو مرتبه باید به مدیر فرستاده شود؛ یک‌بار در ابتدا برای انتقال پردازه از کامپیوتر کارخواه به کامپیوتر کارساز و بار دیگر در انتها برای بازگشت پردازه از کامپیوتر کارساز به کامپیوتر کارخواه برای دریافت نتایج اجرا توسط کامپیوتر کارخواه.

[1] stateless

- در لحظه درخواست اول، لازم است یا تمامی فایل‌های موردنیاز پردازه در طول اجرا توسط پردازه باز شده باشند، یا فایل‌هایی که هنوز باز نشده‌اند در کامپیوتر کارساز موجود باشند. در غیر این صورت، سیستم برون‌سپاری نمی‌تواند نیاز پردازه به فایل‌هایی که این شرایط را ندارند تشخیص دهد و به علت نبود این فایل‌ها در کارساز، اجرای پردازه در کارساز با خطا روبه‌رو خواهد شد.
- قبل از اتمام برنامه، دومین درخواست برون‌سپاری باید به مدیر در کامپیوتر کارساز فرستاده شود تا قبل از بین رفتن پردازه، پردازه به کامپیوتر کارساز باز گردد و نتایج و تغییراتی که این پردازه در حافظه به وجود آورده توسط کامپیوتر کارخواه قابل استفاده باشد. در صورتی که نتایج پردازه در فایل‌هایی ذخیره می‌شود، لازم است قبل از اینکه پردازه فایل‌های مربوط به نتایج را ببندد درخواست برون‌سپاری به مدیر ارسال شود تا این فایل‌ها نیز همراه پردازه به کارخواه فرستاده شوند.

در این فصل به بررسی ابزارهای مورد نیاز برای توسعه سیستم برون‌سپاری پردازه، انتخاب ابزارهای مناسب، و تشریح پیاده‌سازی سیستم برون‌سپاری در سطح پردازه پرداخته شد. در فصل آینده نحوه ارزیابی این سیستم و نتایج بدست آمده از این ارزیابی بررسی می‌شود.

فصل چهارم

ارزیابی و نتایج

ارزیابی و نتایج

در فصل گذشته جزئیات پیاده‌سازی و تعاملات میان اجزا سیستم را بررسی کردیم. در این فصل، با استفاده از برنامه‌های محک[1] عملکرد سیستم را ارزیابی می‌کنیم.

برای اطمینان از صحت عملکرد سیستم، از روش آزمون واحد[2] در حین توسعه استفاده شد. برای انجام این کار، از آزمون‌های جعبه سفید[3] استفاده شد و ایرادات و خطاهای حین اجرا رفع شد تا در نهایت سیستم این آزمون‌ها را با موفقیت به اتمام برساند.

مهم‌ترین پیامد مثبت این پروژه، کوتاه شدن مجموع زمان اجرای وظایف توسط کامپیوترها با منابع سخت‌افزاری محدود است؛ لذا برای ارزیابی این پروژه، زمان اجرای وظایف مورد بررسی قرار خواهد گرفت. برای این کار، زمان اجرای مجموعه‌ای از وظایف مشخص با استفاده از این سیستم و بدون استفاده از آن اندازه‌گیری شده و با یکدیگر مقایسه شده است.

۴-۱-برنامه‌های محک

چهار برنامه محک از مجموعه‌های MiBench [۳۲] و Splash-3 [۳۳] برای ارزیابی این سیستم انتخاب شد. ازآنجاکه برای استفاده از سیستم برون‌سپاری کد منبع برنامه ها باید تغییر کند، متن‌باز بودن این برنامه‌ها ضروری است. با توجه به ملاحظات ذکر شده در بخش۳-۳-۳، کد منبع این چهار برنامه تغییر پیدا کرد تا مناسب اجرا و برون‌سپاری شوند. چهار برنامه محک انتخاب شده عبارت‌اند از:

- **basic math**: این برنامه، مربوط به بخش خودرویی[4] مجموعه MiBench است. در این برنامه محاسبات ریاضی ساده‌ای را انجام می‌دهد که اغلب در پردازنده‌ها پشتیبانی

[1] bemchmark

[2] unit testing

[3] white box testing

[4] automotative

سخت‌افزاری اختصاصی ندارند. به‌عنوان‌مثال، حل معادلات مکعبی[1]، جذر صحیح و تبدیل زاویه‌ها از درجه به رادیان همگی محاسبات ضروری برای محاسبه سرعت جاده یا سایر مقادیر برداری هستند. داده‌های ورودی یک مجموعه ثابت است [۳۲]. برای ارزیابی پروژه، ورودی بزرگ این برنامه در نظر گرفته شده است.

- **qsort**: این برنامه نیز، مربوط به بخش خودرویی مجموعه MiBench است. این برنامه یک آرایه بزرگ از رشته‌ها را با استفاده از الگوریتم معروف مرتب‌سازی سریع[2] به ترتیب صعودی مرتب می‌کند. مرتب‌سازی اطلاعات برای سیستم‌ها مهم است تا اولویت‌ها تعیین شوند، خروجی بهتر تفسیر شود، داده‌ها سازماندهی شوند و زمان اجرای کلی برنامه‌ها کاهش یابد [۳۲]. برای ارزیابی این پروژه، از مجموعه داده بزرگ این برنامه، که شامل مختصات نقاط است، استفاده شده است.
- **ocean**: این برنامه از مجموعه Splash-3 انتخاب شده است و شبیه‌سازی حرکات بزرگ‌مقیاس اقیانوس را بر اساس جریان‌های گردابی و مرزی انجام می‌دهد. در این شبیه‌سازی، نیروی محرکه از تنش باد ناشی از اثرات جوی تأمین می‌شود و تأثیر اصطکاک با دیواره‌ها و کف اقیانوس نیز در نظر گرفته شده است. شبیه‌سازی تا زمانی ادامه دارد تا جریان‌های گردابی و جریان اصلی اقیانوس به تعادل متقابل برسند [۳۳, ۳۴]. برای ارزیابی این پروژه از پیاده‌سازی پیوسته این برنامه استفاده شده است و ابعاد شبکه اقیانوسی، ۲۰۵۰ در ۲۰۵۰ در نظر گرفته شده است. به‌علاوه این شبیه‌سازی در ۸ پردازه موازی انجام می‌شود.
- **water**: این برنامه متعلق به مجموعه Splash-3 است و نیروها و پتانسیل‌هایی که در طول زمان در یک سیستم مولکول‌های آب رخ می‌دهند را ارزیابی می‌کند. محاسبات در طول تعداد مشخصی از گام‌های زمانی که توسط کاربر تعیین می‌شود، با هدف رسیدن به حالت تعادل انجام می‌شود [۳۳, ۳۴]. در این پروژه، از پیاده‌سازی فضایی[3] این برنامه برای ارزیابی

[1] cubic function

[2] quick sort

[3] spatial

استفاده شد و فایل parsec_native به عنوان ورودی برای تعیین پارامتر های برنامه در نظر گرفته شده است. به‌علاوه این ارزیابی در ۸ پردازه موازی انجام می‌شود.

۴-۲-زمان اجرای برنامه‌ها با استفاده از برون‌سپاری

جدول۴-۱، زمان اجرای برنامه‌های محک را با و بدون استفاده از سیستم برون‌سپاری وظیفه، با شرایط مختلف منابع سخت‌افزاری نشان می‌دهد. برای این ارزیابی، روی دو کامپیوتر سیستم‌عامل ubuntu 20.04 به‌صورت ماشین مجازی[1] نصب شد و محدودیت‌های منابع سخت‌افزاری با استفاده از مدیر ماشین مجازی[2] بر روی آنها اعمال شد. مشخصات این دو کامپیوتر در جدول۴-۲ نشان داده شده است. در ستون اول این جدول، وضعیت منابع سخت‌افزاری کامپیوتر کارخواه حین انجام این ارزیابی نشان داده شده است که این وضعیت، در طول ارزیابی در تمام حالات ثابت بوده است. ستون بعدی، برنامه محک اجرا شده را نشان می‌دهد. ستون سوم نتایج را بدون استفاده از برون‌سپاری و ستون چهارم نتایج را با استفاده از برون‌سپاری نشان می‌دهد. زیرستون هر یک ستون‌های سه و چهار، شامل زمان اجرای کلی، زمان اختصاص یافته به هر یک از عملیات‌ها حین اجرا، و وضعیت منابع سخت‌افزاری کامپیوتر کارساز می‌باشد. در این جدول، مواردی که زمان اجرای کلی با استفاده از برون‌سپاری کاهش یافته است با رنگ سبز و در غیر این صورت، با رنگ قرمز مشخص شده است.

[1] virtual machine

[2] virtual machine manager

with offloading									without offloading	test	client setup
total time (ms)	dump (ms) (client)	send (ms) (client to server)	restore (ms) (server)	execution (ms) (server)	dump (ms) (server)	send (ms) (server to client)	restore (ms) (client)	server setup	total time (ms)		
10790	157	1298	536	5013	153	2642	993	2G RAM; 2 CPU cores	8367	basic math	1G RAM; 1 CPU core
7032	29	1240	573	2763	91	1579	755	4G RAM; 8 CPU cores			
8204	82	2343	603	527	143	3697	815	2G RAM; 2 CPU cores	2458	qsort	
7802	50	3373	529	332	73	2670	776	4G RAM; 8 CPU cores			
198964 (~ 3 minutes)	604	6650	692	47941	1348	92953	48543	2G RAM; 2 CPU cores	246318 (~ 4 minutes)	water	
160923 (~ 2 minutes)	533	4029	480	18757	5471	86996	44354	4G RAM; 8 CPU cores			
323091 (~ 5 minutes)	649	4608	786	3945	1886	253184	58093	2G RAM; 2 CPU cores	2371780 (~ 39 minutes)	ocean	
332892 (~ 5 minutes)	591	3732	529	2708	5913	232208	86478	4G RAM; 8 CPU cores			

جدول۴-۱: زمان اجرای برنامه‌های محک.

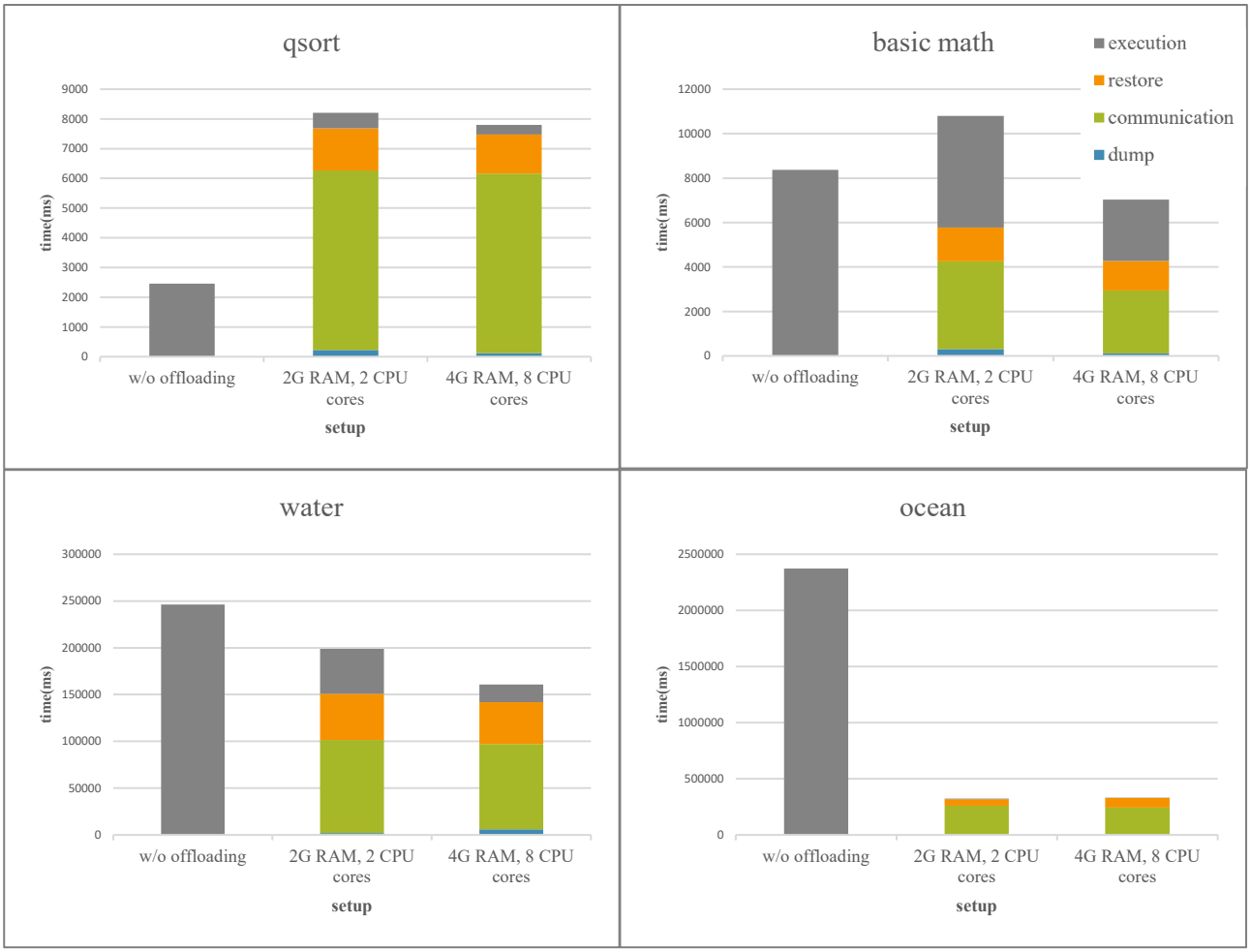
جدول۴-۲: مشخصات کامپیوترهای مورد استفاده در ارزیابی.

client	server	
Intel(R) Core(TM) i7-6700HQ CPU	AMD Ryzen 5 5500U	**processor**
2.6 GHz	4.0 GHz	**processor clock speed**
6 MB	8 MB	**cache**
3.2 MHz	2.4 MHz	**RAM clock speed**

با توجه به جدول۴-۱، اجرای برخی برنامه‌ها با اســتفاده برون‌ســپاری، زمان کمتری نســبت به اجرای آنها بدون برون‌ســپاری دارند. با توجه به جریان وقایع هنگام اسـتفاده از سـیسـتم برون‌سـپاری در شکل۲-۱، رابطه۴-۱، یک رابطه تقریبی برای شرایطی که استفاده از سیستم برون‌سپاری منجر به کوتاه شدن زمان اجرای یک برنامه می‌شود است.

$$T_{execution\ on\ client} > T_{execution\ on\ server} + T_{dump} + T_{restore} + T_{communication} \quad (۱-۴)$$

با توجه به این که برای برون‌سپاری، دو دفعه نیاز به checkpoint، restore، و هماهنگ‌سازی فایل‌ها است، هر کدام از عبارات $T_{communication}$، $T_{restore}$ و $T_{communication}$ مجموع زمان اجرای هر دو دفعه این عملیات ها می‌باشد شکل۴-۱ سهم هر کدام از عبارات رابطه۴-۱ را در دو شرایط مختلف منابع سخت‌افزاری برای چهار برنامه محک نشان می‌دهد.



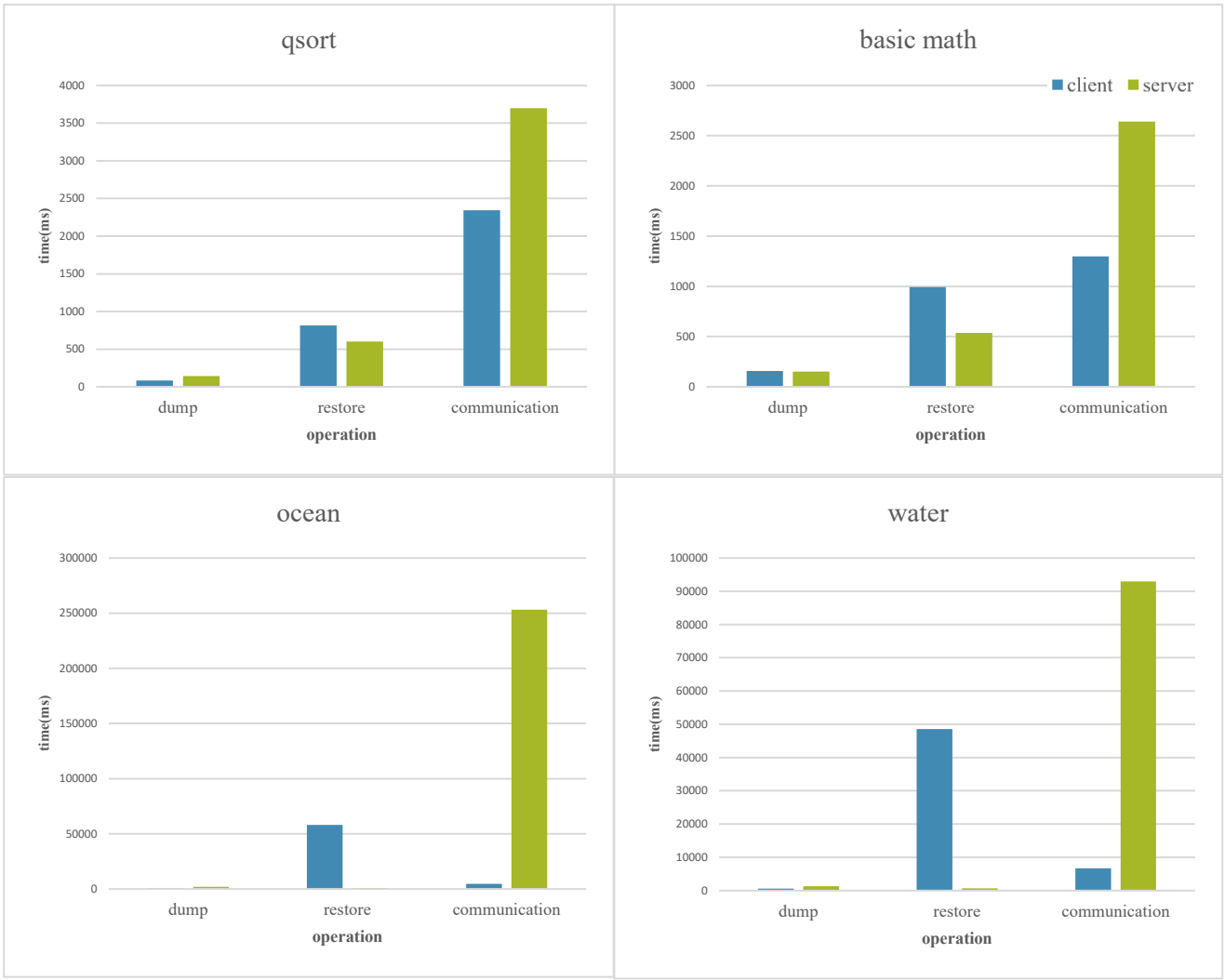
شکل۴-۱: سهم عملیات‌های مختلف حین برون‌سپاری از کل زمان اجرا.

۴-۳-مقایسه سربار زمانی در کارخواه و کارساز

با توجه به رابطه۴-۱، ازآنجاکه هر کدام از عملیات‌های checkpoint، restore و هماهنگ‌سازی فایل‌ها سربار زمانی بر زمان اجرای برنامه اضافه می‌کند؛ در شرایطی که برنامه نیاز زیادی به منابع

سـخت‌افزاری ندارد و زمان اجرای آن کم اسـت، اضـافه شـدن سـربار زمانی عملیات‌های checkpoint، restore و هماهنگ‌سـازی فایل‌ها، بر کاهش زمان اجرای برنامه به جهت اجرا بر روی کامپیوتر با منابع سخت‌افزاری بیشتر غلبه می‌کند. به عبارت دیگر؛ استفاده از برون‌سپاری در مواردی که برنامه نیاز زیادی به منابع سخت‌افزاری ندارد می‌تواند منجر به بیشتر شدن زمان اجرای نهایی برنامه شود.

همان‌طور که پیش‌تر اشـاره شـد، برای برون‌سـپاری صـحیح، روی هر برنامه عملیات‌های checkpoint، restore و هماهنگ‌سـازی فایل‌ها یک‌بار در کامپیوتر کارخواه انجام می‌شـود و یک‌بار در کامپیوتر کارساز. شکل۴-۲ زمان اجرای این عملیات‌ها را روی کامپیوتر کارخواه و کارساز مقایسه می‌کند.



شکل۴-۲: مقایسه زمان اجرای عملیات‌های مختلف بر روی کامپیوترهای کارساز و کارخواه.

با توجه به شـکل۴-۲ در اغلب موارد، زمان اجرای عملیات‌های checkpoint و هماهنگ‌سـازی فایل‌ها بر روی کامپیوتر کارسـاز، بیشـتر از زمان اجرای آن‌ها در کامپیوتر کارخواه اسـت. به‌علاوه زمان اجرای عملیات restore، در کامپیوتر کارخواه بیشـتر از کامپیوتر کارسـاز اسـت. یکی از دلایل اصـلی این اختلاف زمان اجرا، تفاوت حجم فضـای حافظه اشـغال شـده توسـط پردازه‌ها در این دو حالت اسـت. جدول۴-۳ حجم مجموعه فایل‌های تصـویر یک پردازه را در کامپیوتر کارخواه، که حاصـل اولین مرتبه checkpoint در ابتدای برنامه به جهت برون‌سـپاری پردازه به کامپیوتر کارسـاز اسـت، و در کامپیوتر کارسـاز، که حاصـل دومین مرتبه checkpoint در انتهای برنامه به جهت بازگشـت پردازه به کامپیوتر کارخواه برای استفاده از نتایج آن در کامپیوتر کارخواه است را نشان می‌دهد.

جدول۴-۳: مقایسه مجموع حجم فایل‌های تصویر در کامپیوترهای کارخواه و کارساز.

server	client	
22 MB	14 MB	**basic math**
25 MB	19 MB	**qsort**
1.2 GB	29 MB	**water**
1.9 GB	23 MB	**ocean**

با توجه به اینکه حجم فایل‌های تصویر متناسب با حجم حافظه اشغال شده توسط پردازه است [۱۰]، مجموع حجم فایل‌های تصویر، معیار مناسبی برای مقایسه حجم حافظه اشغال شده توسط پردازه است. با توجه به جدول۴-۳، مجموع حجم فایل‌های تصـویر، و متناسـب با آن حجم حافظه اشـغال شـده توسـط پردازه، در هنگام گرفتن checkpoint در کارسـاز، بیشـتر از هنگام checkpoint گرفتن در کارخواه است.

دلیل این اختلاف این اسـت که در هنگام گرفتن checkpoint در کامپیوتر کارخواه، برنامه در ابتدای اجرای خود قرار دارد. با اجرای برنامه و رسـیدن به انتهای آن، حجم فضـای حافظه اشـغال شـده توسط پردازه افزایش می‌یابد. در نتیجه در هنگام گرفتن checkpoint در کامپیوتر کارساز، که برنامه در

انتهای اجرای خود قرار دارد، حجم حافظه اشغال شده توسط آن بیشتر است. در نتیجه مجموع حجم فایل‌های تصویر بیشتر است.

در نهایت با توجه به اینکه حجم فایل‌های تصویر و زمان اجرای عملیات‌های checkpoint و restore، متناسب با حجم حافظه اشغال شده توسط پردازه است [۱۰]، سربار زمانی عملیات های checkpoint، restore و هماهنگ‌سازی فایل‌ها در مرتبه دوم انجام آنها بیشتر است. با توجه به شکل۴-۲، این اختلاف برای برنامه‌های ocean و water که برنامه‌های سنگین تری نسبت به qsort و basic math هستند، نمایان‌تر است.

فصل پنجم

جمع‌بندی و نتیجه‌گیری و پیشنهادها

جمع‌بندی و نتیجه‌گیری و پیشنهادها

در این پروژه، یک سیستم برون‌سپاری وظایف با رویکرد برون‌سپاری در سطح پردازه توسعه داده شد. بر اساس نتایج حاصل از ارزیابی سیستم با استفاده از برنامه‌های محک استاندارد، این سیستم در مواردی که برنامه سنگین است و احتیاج زیادی به منابع سخت‌افزاری دارد، کارایی قابل توجهی دارد و می‌تواند زمان اجرای نهایی برنامه را به طور چشمگیری کاهش دهد. با این حال استفاده از این سیستم با محدودیت‌هایی همراه است و پردازه‌های را که دارای شرایط خاصی هستند را نمی‌تواند به طور صحیح برون‌سپاری کند.

در این پروژه از دو ابزار CRIU و rsync استفاده شده است. هر کدام از این دو ابزار محدودیت‌هایی دارند که در پژوهش‌های آتی می‌تواند مورد بررسی و بهبود قرار گیرد.

۵-۱-محدودیت‌های ناشی از CRIU و پیشنهادها برای رفع آن‌ها

در بخش۳-۱-۳ به محدودیت‌های CRIU اشاره شده است. در مواردی که فایل‌های مورد استفاده توسط پردازه در دو کامپیوتر متفاوت باشند یا در صورتی که ابزارها و کتابخانه‌های مورد استفاده توسط پردازه در دو کامپیوتر از نسخه‌های متفاوتی باشند، پردازه‌ها به طور صحیح نمی‌توانند برون‌سپاری شوند.

به‌علاوه، بازگشت پردازه‌ها از کامپیوتر کارساز به کامپیوتر کارخواه دارای سربار زمانی و شبکه قابل توجهی است؛ در حالی که این بازگشت در انتهای اجرای برنامه و تنها به منظور استفاده از نتایج اجرای برنامه در کامپیوتر کارخواه انجام می‌شود. در پژوهش‌های آینده می‌توان بر برطرف کردن محدودیت‌های CRIU یا کاهش سربار استفاده از آن تمرکز کرد تا کارایی سیستم افزایش یابد.

۵-۲-محدودیت‌های ناشی از rsync و پیشنهادها برای رفع آن‌ها

استفاده از rsync به جهت برخورداری از الگوریتمی سریع برای هماهنگ‌سازی فایل‌ها و سایر ویژگی‌های این ابزار مانند فشرده‌سازی داده‌های ارسالی، سربار ارتباطات شبکه را کاهش می‌دهد. با این

وجود ازآنجاکه این ابزار بر روی ارتباط ssh اجرا می‌شود، سربار رمزگذاری به آن اضافه می‌شود. با حذف این رمزگذاری می‌توان سرعت هماهنگ‌سازی فایل‌ها را کاهش داد.

۵-۳-پیشنهادها برای توسعه سیستم برون‌سپاری وظیفه

در این پروژه اعلام درخواست برون‌سپاری توسط پردازه انجام می‌شود؛ لذا دستورات مربوط انجام به این کار باید پیش از اجرای برنامه توسط توسعه دهنده در برنامه تعبیه شود. با توجه به اینکه اطلاعات آماری مفیدی از شرایط اجرای یک پردازه در حین اجرا در دسترس سیستم‌عامل است، تشخیص و انتخاب پردازه‌هایی که می‌توانند برون‌سپاری شوند می‌تواند از وظایف توسعه دهنده خارج شود. به این منظور، لازم است یک الگوریتم انتخاب برای تشخیص و برون‌سپاری پردازه‌ها توسعه داده شود. در پژوهش‌های آینده می‌توان بر روی طراحی چنین الگوریتمی تمرکز کرد. لازم به ذکر است توسعه چنین الگوریتمی با توجه به محدودیت سیستم برای برون‌سپاری برخی پردازه‌ها و اهمیت انتخاب موقعیت مناسب برای برون‌سپاری یک پردازه که در بخش‌های ۳-۱-۳ و ۳-۳-۳ به آن اشاره شد، چالش‌های فراوانی خواهد داشت.

به‌علاوه از آنجا معماری سیستم برون‌سپاری وظیفه در کامپیوترهای کارخواه و کارساز به طور متقارن است، به‌سادگی می‌توان این سیستم را به‌گونه‌ای تغییر داد تا مناسب برون‌سپاری چندسطحی باشد، به‌طوری که پس از برون‌سپاری پردازه به یک کامپیوتر کارساز، در صورتی که بار زیادی بر روی کامپیوتر کارساز باشد و برنامه‌های زیادی بر روی این کامپیوتر مشغول اجرا باشند؛ به‌طوری که پردازه‌های برون‌سپاری شده نتواند به اندازه کافی از منابع سخت‌افزاری استفاده کند، یا در صورتی که بخشی از پردازه‌های برون‌سپاری شده احتیاج به منابع سخت‌افزاری داشته باشند که در این کامپیوتر کارساز موجود نباشد، این کامپیوتر می‌تواند بخشی از پردازه‌ها را به کامپیوتر کارساز دیگری منتقل کند. در این حالت، کامپیوتر کارساز اول به عنوان کارخواه، و کامپیوتر کارساز دوم به عنوان کارساز می‌توانند عمل کنند تا با کمترین تغییرات در معماری سیستم، برون‌سپاری به‌صورت چندسطحی انجام شود.

۵-۴-استفاده از سیستم برون‌سپاری وظیفه بر روی سایر سیستم‌های عامل

در این پروژه، سیستم برون‌سپاری وظیفه جهت استفاده بر روی سیستم‌عامل لینوکس توسعه داده شده است. در نتیجه این سیستم نمی‌تواند بر روی سایر سیستم‌های عامل استفاده شود. با این وجود مفاهیم و طراحی کلی سیستم برون‌سپاری وظیفه با رویکرد مهاجرت پردازه، بر روی سایر سیستم‌های عامل نیز مشابه این سیستم خواهد بود. چالش اصلی، استفاده از ابزار مناسب جهت انجام عملیات‌های checkpoint و restore است. تا کنون ابزار شناخته شده‌ای که قادر به انجام عملیات‌های لازم برای مهاجرت پردازه باشد برای سیستم‌عامل‌های mac و windows توسعه داده نشده است. به‌علاوه، ممکن است برای توسعه چنین ابزاری برای این دو سیستم‌عامل، نیاز به انجام تغییرات در هسته سیستم‌عامل باشد که کار توسعه این ابزار را دشوار می‌کند.

منابع و مراجع

[1] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, “Edge cloud offloading algorithms: Issues, methods, and perspectives,” *ACM Computing Surveys (CSUR),* vol. 52, no. 1, pp. 1-23, 2019.

[2] A. Yousafzai, I. Yaqoob, M. Imran, A. Gani, and R. M. Noor, “Process migration-based computational offloading framework for IoT-supported mobile edge/cloud computing,” *IEEE internet of things journal,* vol. 7, no. 5, pp. 4171-4182, 2019.

[3] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, “A survey and taxonomy on task offloading for edge-cloud computing,” *IEEE Access,* vol. 8, pp. 186080-186101, 2020.

[4] M. Cui, S. Zhong, B. Li, X. Chen, and K. Huang, “Offloading autonomous driving services via edge computing,” *IEEE Internet of Things Journal,* vol ,۷ .no. 10, pp. 10535-10547, 2020.

[5] Z. Xiao, J. Shu, H. Jiang, G. Min, H. Chen, and Z. Han, “Perception task offloading with collaborative computation for autonomous driving,” *IEEE Journal on Selected Areas in Communications,* vol. 41, no. 2, pp. 457-473.۲۰۲۲ ,

[6] X. Chen, and G. Liu, “Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks,” *IEEE Internet of Things Journal,* vol. 8, no. 13, pp. 10843-10856, 2021.

[7] S. Imai ,and C. A. Varela, "Light-weight adaptive task offloading from smartphones to nearby computational resources." pp. 146-152.

[8] P. Chen, L. Luo, D. Guo, X. Luo, X. Li, and Y. Sun, “Secure Task Offloading for Rural Area Surveillance Based on UAV-UGV Collaborations,” *IEEE Transactions on Vehicular Technology*, 2023.

[9] M. T. Chung, J. Weidendorfer, K. Fürlinger, and D. Kranzlmüller, "Proactive task offloading for load balancing in iterative applications." pp. 263-275.

[10] A. Tošić, “Run-time application migration using checkpoint/restore in userspace,” *arXiv preprint arXiv:2307.12113*, 2023.

[11] A. Yousafzai, A. Gani, R. M. Noor, A. Naveed, R. W. Ahmad, and V. Chang, “Computational offloading mechanism for native and android runtime based mobile applications,” *Journal of Systems and Software,* vol. 121, pp. 28-39, 2016.

[12] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud." pp. 784-791.

[13] B.-D. Lee, "A framework for seamless execution of mobile applications in the cloud," *Recent Advances in Computer Science and Information Engineering: Volume 3*, pp. 145-153: Springer, 2012.

[14] R. K. Ma, and C.-L. Wang, "Lightweight application-level task migration for mobile cloud computing." pp. 550-557.

[15] T. Verbelen, T. Stevens, P. Simoens, F. De Turck, and B. Dhoedt, “Dynamic deployment and quality adaptation for mobile augmented reality applications,” *Journal of Systems and Software,* vol. 84, no. 11, pp. 1871-1882, 2011.

[16] E. Y. Chen, and M. Itoh, "Virtual smartphone over IP." pp. 1-6.

[17] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: versatile protection for smartphones." pp. 347-356.

[18] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "{COMET}: Code offload by migrating execution transparently." pp. 93-106.

[19] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, “Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing,” *IEEE Journal on Selected Areas in Communications,* vol. 37, no. 3, pp. 668-682, 2019.

[20] C. Press, “IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things,” 2017.

[21] "CRIU official website," https://criu.org./

[22] "DMTCP official website," https://dmtcp.sourceforge.io./

[23] J. Duell, “The design and implementation of berkeley lab's linux checkpoint/restart,” 2005.

[24] "OpenVZ official website," https://openvz.org./

[25] "SCP documentation," https://www.man7.org/linux/man-pages/man1/scp.1.html.

[26] "Syncthing documentation," https://docs.syncthing.net./

[27] "FreeFileSync official website," https://freefilesync.org./

[28] "Rclone official website," https://rclone.org./

[29] "rsync official website," https://rsync.samba.org./

[30] Y .Chen, "Checkpoint and restore of micro-service in docker containers." pp. 915-918.

[31] A. Tridgell, and P. Mackerras, “The rsync algorithm,” 1996.

[32] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free ,commercially representative embedded benchmark suite." pp. 3-14.

[33] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research." pp. 101-111.

[34] J. P. Singh, W.-D. Weber, and A. Gupta, “SPLASH: Stanford parallel applications for shared-memory,” *ACM SIGARCH Computer Architecture News,* vol. 20, no. 1, pp. 5-44, 1992.

Abstract

With the development of edge cloud infrastructures and the Internet of Things, and the increasing demands of applications for hardware resources, task offloading has gained more attention than ever before. Various approaches for task offloading have been proposed in the past, each of which, due to certain limitations, failed to achieve expected development. In this project, a task offloading system with a process-level offloading approach (using CRIU tool) has been developed to address some of the limitations of previous methods. In this approach, processes that need to be offloaded are suspended on the source computer, transferred to the destination computer, executed there and eventually returned to the source computer upon completion. This system has demonstrated promissing performance in evaluations using standard benchmark programs, and in the best case, the overall execution time of one of the benchmark programs was reduced by 87% using this system. By addressing some of the limitations of this offloading system, proper development can be expected.

Key Words: task offloading, process migration, checkpoint, restore, edge cloud computing

Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Enfineering

Bachelor Thesis

OS-level task offloading between two computers using process migration mechanism

By
Mohammad Ali Moghimi

Supervisor
Dr. Hamid Reza Zarandi

July 2024