گزارش پروژه بی درنگ

مهدیار احمدی زاده - ۹۹۱۷۰۳۳۷
بردیا رضایی کلانتری- ۹۹۱۷۰۴۵۶

پروژه شامل فایل‌های زیر است که هرکدام بخشی از عملکرد زمان‌بندی بی‌درنگ را مدیریت می‌کنند:

- runner.py: نقطه شروع اجرای پروژه است.
- core.py: تعریف هسته‌ها و ویژگی‌های آن‌ها.
- cuckoo2.py: پیاده‌سازی الگوریتم جستجوی Cuckoo برای زمان‌بندی.
- job.py: تعریف وظایف کلی و وظایف فرعی در طول هایپرپریود.
- task.py: تعریف Taskها و ویژگی‌هایشان.
- uunifast.py: تولید وظایف با استفاده از الگوریتم UUniFast.
- output_generator.py: تولید خروجی‌های پروژه مثل فایل‌های گزارش.
- mapper.py: نگاشت بین داده‌ها.
- sbti.py: ابزار یا الگوریتم دیگری برای زمان‌بندی.
- README.md: توضیحات کلی پروژه.

در کل چون اجرای پروژه هم زمان فاز ۱ و فاز ۲ خیلی سنگین بود من خروجی هر بخش در یک فایل json ذخیره کردیم و هر مرحله جدا اجرا میکردیم که کد های این بخش در runner.py می باشد.

فایل runner

runner_generate_task

برای هر ترکیب از تعداد هسته و بازده، ۵۰ تسک تولید می‌کند. تسک‌ها شامل ویژگی‌هایی مثل دوره، اجرای مورد نیاز و ددلاین هستند. سپس اطلاعات این تسک‌ها را در فایل‌های JSON در مسیر ./tasks/ ذخیره می‌کند. هدف آماده‌سازی ورودی برای فازهای بعدی زمان‌بندی است.

phase1

تسک‌های تولیدشده را از فایل می‌خواند و آن‌ها را به سخت (hard) و نرم (soft) تقسیم می‌کند. سپس با الگوریتم WFD تسک‌های سخت را روی هسته‌ها نگاشت کرده و زمان‌بندی می‌کند. تسک‌های نرم با استفاده از الگوریتم SBTI زمان‌بندی می‌شوند. نتیجه شامل زمان‌بندی، نگاشت تسک‌ها و بهره‌وری کلی در فایل JSON ذخیره می‌شود.

phase2_hardtask

تسک‌ها را خوانده و تسک‌های سخت را با استفاده از الگوریتم SFLA روی هسته‌ها نگاشت می‌کند. خروجی شامل اطلاعات کامل زمان‌بندی و نگاشت تسک‌هاست. فایل خروجی در مسیر ./phase2/ ذخیره می‌شود.

phase2_softask

تسک‌های نرم و هسته‌ها را از خروجی فاز ۲ می‌خواند.با الگوریتم Cuckoo تلاش می‌کند تسک‌های نرم را روی هسته‌ها زمان‌بندی کند. تسک‌های پذیرفته‌شده به برنامه زمان‌بندی اضافه شده و بقیه به عنوان drop ثبت می‌شوند. در پایان فایل JSON بروزرسانی شده ذخیره می‌شود.

create_output

با استفاده از توابع last_exec و power از ماژول output_generator خروجی نهایی تولید می‌کند. هدف این مرحله پردازش نهایی برای ارزیابی یا نمایش داده‌هاست. این تابع به صورت خلاصه تجزیه و تحلیل کلی انجام می‌دهد. مناسب برای گزارش‌گیری است.

حالا من در اینجا از چند کلاس task,job,core استفاده کردم که **job** در واقع همون تسک ها هستند تا هایپرپریود و هر هسته یه سری تسک دارند که باید اجرا شوند.

فایل core

Add_task

تسک مشخص‌شده را به لیست تسک‌های این هسته اضافه می‌کند.

total_load

مجموع زمان اجرای تسک‌های زمان‌بندی‌شده روی این هسته را محاسبه می‌کند.

calculate_hyperperiod

هایپرپریود را با محاسبه کم‌ترین مضرب مشترک (LCM) از دوره تسک‌های اختصاص‌یافته به این هسته به‌دست می‌آورد.

generate_jobs

برای هر تسک، jobهای دوره‌ای تا هایپرپریود تولید می‌کند که شامل زمان آزادسازی، ددلاین و اجرای لازم هستند.

get_slack

فاصله‌های زمانی بین jobهای زمان‌بندی‌شده (فضای خالی یا اسلک) را محاسبه و ذخیره می‌کند.
در پایان نیز مقدار زمان رسیدن به پایان زمان‌بندی را بازمی‌گرداند.

edf_schedule

با استفاده از الگوریتم EDF (Earliest Deadline First)، jobهای تسک‌ها را در طول هایپرپریود روی این هسته زمان‌بندی می‌کند.

get_earliest_start_time

اولین زمان ممکن برای اجرای یک تسک در اسلک‌های موجود را بازمی‌گرداند، اگر فضای کافی برای اجرای آن وجود داشته باشد.

get_slack_intervals

فاصله‌های زمانی خالی بین اجرای تسک‌ها (اسلک‌ها) را به‌صورت لیستی از بازه‌ها برمی‌گرداند.

schedule_soft_task

تسک نرم را در بازه مشخص‌شده روی هسته زمان‌بندی می‌کند و زمان‌بندی نهایی را به‌صورت مرتب‌شده ذخیره می‌نماید.

فایل task

هم نکته خواصی نداره فقط to_dict مهمه جون برای ذخیره تسک ها در فایل ها از اون استفاده میکردم.

فایل uunifast

normalize_array

آرایه‌ای از مقدارهای utilization را می‌گیرد و نرمال‌سازی می‌کند.کم‌ترین مقدار را به میانگین با بیشترین مقدار تبدیل کرده و بیشترین مقدار را دو برابر مقدار جدید کمینه می‌کند. هدف آن کاهش مقدارهای بیش‌ازحد بزرگ (≤1) در uunifast است. بازگشتی از آرایه‌ی نرمال‌شده را می‌دهد.

uunifast

الگوریتم UUniFast برای تولید n مقدار utilization که جمعشان برابر u_total باشد. استفاده از توزیع یکنواخت (random.uniform) برای ایجاد پراکندگی بین مقادیر. در صورتی که یکی از مقادیر ≤1 شود، با normalize_array اصلاح می‌شود.در نهایت لیستی از utilizationهای مجاز (زیر ۱) باز می‌گرداند.

generate_tasks

با استفاده از n ،uunifast تسک با مجموع utilization برابر u_total * m تولید می‌کند. ۳۰ تسک اول به‌صورت سخت (hard) و بقیه نرم (soft) در نظر گرفته می‌شوند. برای هر تسک، execution به صورت تصادفی و period متناسب با utilization محاسبه می‌شود. در نهایت لیستی از اشیای کلاس Task برمی‌گرداند که می‌توان در زمان‌بندی استفاده کرد.

فایل sbti

1. برای هر تسک نرم در soft_tasks:
   - ابتدا فرض می‌شود که هنوز زمان‌بندی نشده (scheduled = False).
2. سپس روی همه هسته‌ها (cores) تکرار می‌کند:
   - از هر هسته، بازه‌های زمانی خالی (slack intervals) را با get_slack_intervals() می‌گیرد.
3. اگر طول یک اسلک (یعنی interval[1] - interval[0]) بزرگ‌تر یا مساوی با زمان اجرای تسک باشد:
   - آن تسک در همان بازه با schedule_soft_task(task, interval) زمان‌بندی می‌شود.
   - scheduled = True شده و بقیه حلقه‌ها شکسته می‌شوند.

فایل mapper

wfd_mapping

این تابع وظیفه دارد تسک‌های سخت را بین هسته‌ها توزیع کند. تسک‌ها ابتدا بر اساس بهره‌وری (execution/period) به‌صورت نزولی مرتب می‌شوند.سپس هر تسک سخت به هسته‌ای داده می‌شود که کمترین مجموع بهره‌وری فعلی را دارد. در نهایت آرایه‌ی cores با تسک‌های تخصیص‌یافته بازگردانده می‌شود.

sfla

الگوریتم SFLA برای بهینه‌سازی تخصیص تسک‌های سخت به هسته‌ها استفاده می‌شود. ابتدا مجموعه‌ای از جواب‌های تصادفی (frogs) تولید می‌شود. در هر تکرار، frogs به گروه‌های کوچک‌تر تقسیم شده و ضعیف‌ترین جواب در هر گروه به سمت بهترین جواب حرکت می‌کند. در پایان، بهترین frog (الگوی تخصیص بهینه) انتخاب و بازگردانده می‌شود.

create_frog

یک frog (جواب کاندید) ایجاد می‌کند که نشان می‌دهد هر تسک به کدام هسته اختصاص یافته. به‌ازای هر تسک، یک عدد تصادفی بین 0 و (تعداد هسته‌ها - 1) انتخاب می‌شود.خروجی: لیستی از اندیس هسته‌ها به طول تعداد تسک‌ها. درون sfla برای ایجاد جمعیت اولیه استفاده می‌شود.

evaluate_frog

بهره‌وری هر frog را بر اساس تخصیص فعلی تسک‌ها به هسته‌ها محاسبه می‌کند.
هر تسک با توجه به مقدار بهره‌وری‌اش به هسته متناظر جمع می‌شود. مقدار بازگشتی، بیشترین بهره‌وری بین همه هسته‌هاست (معیار کیفیت). هدف الگوریتم: کمینه کردن این مقدار.

move_frog

frog ضعیف (بدترین) را به سمت frog قوی‌تر (بهترین) حرکت می‌دهد. برای هر تسک، با احتمال ۰.۵ تخصیص آن را مثل frog قوی‌تر تنظیم می‌کند. این تابع باعث بهبود جواب‌های ضعیف می‌شود. خروجی: frog جدید با تغییرات احتمالی در تخصیص.

فایل cuckoo

cuckoo

الگوریتم بهینه‌سازی Cuckoo Search را برای زمان‌بندی وظایف نرم روی چند هسته اجرا می‌کند.هر لانه (nest) یک راه‌حل است که نشان می‌دهد هر تسک به کدام هسته تخصیص داده شده است. با استفاده از پرش‌های Levy و جایگزینی لانه‌های ضعیف، سعی می‌کند بهترین تخصیص را پیدا کند. در پایان، اگر نیاز باشد، تسک‌ها را به بخش‌های کوچک‌تر تقسیم کرده و روی هسته‌ها برنامه‌ریزی می‌کند.

Fitness

مقدار بار (load) هر هسته را بر اساس تخصیص تسک‌ها محاسبه می‌کند. اگر هیچ فضای خالی برای اجرای تسک پیدا نشود، مقدار بی‌نهایت (∞) بازمی‌گرداند.
در غیر این صورت، بیشترین بار بین هسته‌ها را برمی‌گرداند. هدف، حداقل‌سازی بیشترین بار است.

levy_flight

یک پرش Levy تولید می‌کند که تغییرات تصادفی برای جهش راه‌حل‌ها در الگوریتم Cuckoo ایجاد می‌کند. این تغییرات باعث جستجوی فضای حل می‌شود. فرمول استاندارد توزیع Levy برای تولید گام‌ها استفاده شده است. خروجی: برداری از گام‌ها برای هر تسک.

split_task

تسکی با زمان طولانی را به تکه‌های کوچک‌تر با اندازه‌های ۳، ۲ یا ۱ تقسیم می‌کند. از قطعات بزرگ‌تر شروع می‌کند تا تعداد بخش‌ها کمتر باشد. هدف این است که تسک‌هایی که به طور کامل قابل زمان‌بندی نیستند، به‌صورت بخش‌بخش اجرا شوند. خروجی: لیستی از اندازه‌های بخش‌های تقسیم شده.

find_earliest_slot

زودترین زمانی را در هسته مشخص می‌کند که می‌توان تسکی با task_duration اجرا کرد بدون برخورد با برنامه‌های قبلی. اطلاعات بازه‌های اشغال‌شده از schedule"]"core"] خوانده می‌شود. تا زمانی که زمان مناسب پیدا نشود، جلو می‌رود. اگر هیچ بازه‌ی مناسبی تا پایان هایپرپریود (یا یک مقدار پیش‌فرض) پیدا نشود، None بازمی‌گرداند.

فایل generate_output

Last_exec

این تابع مدت زمان نهایی اجرای تسک‌ها (آخرین زمان پایان) را از فایل‌های نتایج فاز ۱ و ۲ برای پیکربندی‌های مختلف هسته‌ها و کارایی‌ها استخراج می‌کند. از فایل‌های JSON داده می‌خواند و بزرگ‌ترین مقدار end را برای هر ترکیب استخراج می‌کند. داده‌ها در دو لیست (y1 و y2) ذخیره شده و سپس روی یک نمودار دوبخشی رسم می‌شوند. نمودار حاصل تفاوت زمان اجرای نهایی را در فازهای مختلف مقایسه می‌کند. نتیجه در فایل تصویری last_exec.png ذخیره می‌شود.

Power

تابع اصلی تحلیل مصرف انرژی و کیفیت خدمات (QoS) سیستم است. برای هر فایل خروجی از فاز ۱ و ۲، توان مصرفی هر هسته و QoS هر تسک نرم را محاسبه می‌کند. نتایج با استفاده از چند تابع داخلی محاسبه و به‌صورت JSON در مسیر ./output/ ذخیره می‌شوند. از پارامترهایی مانند ولتاژ، فرکانس، بهره‌برداری و ... برای تخمین انرژی استفاده می‌کند. QoS نیز بر اساس رسیدن یا نرسیدن به موعد تحویل (deadline) هر تسک محاسبه می‌شود. خروجی شامل میانگین QoS سیستم و توان کل مصرفی است.

calculate_core_powere

توان مصرفی یک هسته را با استفاده از مدل توان پویا (dynamic) و ایستا (static) محاسبه می‌کند. ورودی شامل پارامترهایی مانند ولتاژ، فرکانس، بهره‌برداری (utilization) و ... است. توان پویا و ایستا طبق همون فرمول که از ta گرفتم حساب میشه خروجی شامل جزئیات هر دو نوع توان، بهره‌برداری و اطلاعات شناسایی هسته است. مقدار برگشتی برای استفاده در تحلیل مصرف انرژی سیستم استفاده می‌شود.

calculate_task_qos

برای هر تسک در زمان‌بندی، QoS را بر اساس میزان تاخیر نسبت به deadline محاسبه می‌کند. اگر پایان یک اجرا قبل از deadline باشد، امتیاز QoS برابر با ۱ خواهد بود. اگر کمی بعد از deadline تمام شود، QoS بین ۰ تا ۱ محاسبه می‌شود. اگر خیلی دیرتر تمام شود، QoS صفر و به عنوان miss در نظر گرفته می‌شود. برای هر تسک، نرخ از دست رفتن deadline و متوسط QoS را ارائه می‌دهد. داده‌های ورودی شامل زمان‌بندی و تعاریف تسک‌ها هستند.

calculate_system_metric

ترکیبی از محاسبات انرژی و QoS برای کل سیستم است. ابتدا تعاریف تسک‌ها را از داده‌های JSON استخراج می‌کند. سپس برای هر هسته، مصرف انرژی و QoS را محاسبه می‌کند. میانگین QoS در سطح سیستم و توان مصرفی کل نیز محاسبه می‌شود. نتایج به‌صورت دیکشنری شامل بخش‌های "power_metrics" و "qos_metrics" بازمی‌گردد. این خروجی در تابع power() ذخیره می‌شود.