Journal of Information and communication Technology in policing (JICTP)
Journal Homepage: pitc.jrl.police.ir

Vol. 2, No.3 (7), Autumn 2021

Provide a method for Scheduling of dependent and periodic tasks in real-time mixed-criticality systems with multi-core architecture with the aim of preemption reduction

Received: 12 October 2021
Accepted: 13 November 2021
Article type: Research Article
PP: 53-67

DOI:
10.22034/pitc.2021.210851.1091

Fatemeh Azad
Faculty of Electrical and Computer Engineering, Qom University of Technology, Qom, Iran
azad.f@qut.ac.ir

Morteza Mohajjel kafshdooz
Faculty of computer engineering group, Qom University of Technology
mohajjel@qut.ac.ir

AbdolReza Rasouli Kenari
Faculty of computer engineering group, Qom University of Technology
rasouli@qut.ac.ir

Abstract
Today, due to the tendency to integrate different parts of a real-time system with different criticality levels, the concept of mixed-criticality systems has been considered. Applications of these systems include UAVs and smart police devices, in which different tasks of different importance are integrated together on a common platform. In order to provide the processing power required by mixed-criticality systems, multi-core architectures are a good option. One of the main challenges in multi-core architectures is task scheduling. Many researches in the field of task scheduling in mixed-criticality systems with multi-core architecture are assuming tasks are independent, but in reality, we are faced with many tasks that have a data dependency between them. Another aspect of task scheduling in lesser-known multi-core mixed-criticality systems is the reduction and control of the number of preemptions, which reduces time overhead during task execution.
In this research, we schedule dependent tasks in a multi-core architecture in such a way that, firstly, the proposed schedule is correct and satisfies the limitations of the system, and secondly, it reduces the number of preemptions when assigning tasks to cores. Finally, the proposed scheduling algorithm is tested on a sample UAV and random tasks and then the scheduling rate and number of preemptions are compared. The results show that the proposed algorithm reduces the number of preemptions by about 96% without significantly changing the scalability rate.

Keywords: real-time system mixed-criticality system multi-core dependent tasks scheduling preemption reduction

نشریه عملی "فناوری اطلاعات و ارتباطات انتظامی"

سامانه نشریات: pitc.jrl.police.ir

سال دوم- شماره سوم(پیاپی ۷)– پاییز ۱۴۰۰

ارائه روشی برای زمان‌بندی وظایف متناوب و وابسته در سامانه‌های بی‌درنگ بحرانی–مختلط چندهسته‌ای با هدف کاهش تعداد قبضه‌ها

دریافت: ۱۴۰۰/۰۷/۲۰

پذیرش: ۱۴۰۰/۰۸/۲۲

نوع مقاله: پژوهشی

صص: ۶۷-۵۳

شناسه دیجیتال(doi).

10.22034/pitc.2021.210851.1091

فاطمه آزاد

دانشجوی کارشناسی ارشد مهندسی کامپیوتر، دانشکده برق و کامپیوتر- دانشگاه صنعتی قم- قم - ایران

azad.f@qut.ac.ir

مرتضی محجل کفشدوز

استادیار، دانشکده برق و کامپیوتر- دانشگاه صنعتی قم- قم - ایران (نویسنده مسئول)

mohajjel@qut.ac.ir

عبدالرضا رسولی کناری

استادیار، دانشکده برق و کامپیوتر- دانشگاه صنعتی قم- قم - ایران

rasouli@qut.ac.ir

چکیده

امروزه به علت تمایل به یکپارچه‌سازی بخش‌های مختلف سامانه‌های بی‌درنگ با بحرانیت‌های متفاوت در صنعت، سامانه‌های بحرانی-مختلط مورد توجه قرار گرفته‌اند. از جمله کاربردهای این سامانه‌ها می‌توان به پهپادها و ابزارهای پلیس هوشمند اشاره کرد که در آن‌ها، وظایف مختلف با میزان اهمیت متفاوت در یک پلتفرم مشترک جمع‌آوری شده‌اند. به منظور فراهم کردن توان پردازشی مورد نیاز سامانه‌های بحرانی-مختلط، معماری‌های چندهسته‌ای گزینه مناسبی به شمار می‌آیند. یکی از چالش‌های اصلی در معماری‌های چند هسته‌ای، زمان‌بندی وظایف است. بسیاری از پژوهش‌های انجام شده در حوزه زمان‌بندی وظایف در سامانه‌های بحرانی-مختلط چندهسته‌ای به زمان‌بندی وظایف مستقل پرداخته‌اند. ولی در واقعیت با بسیاری از وظایف روبرو هستیم که وابستگی داده‌ای بین آن‌ها وجود دارد و تا زمانی‌که اجرای یک وظیفه به اتمام نرسد امکان شروع وظیفه وابسته به آن وجود ندارد. جنبه دیگر از زمان‌بندی وظایف در سامانه‌های بحرانی-مختلط که کمتر به آن پرداخته شده کاهش تعداد قبضه‌های وظایف است که موجب کاهش سربار زمانی در حین اجرای وظایف می‌شود. در این پژوهش به زمان‌بندی وظایف وابسته و متناوب در سامانه‌های بحرانی-مختلط چندهسته‌ای خواهیم پرداخت به گونه‌ای که اولا زمان‌بندی ارائه شده محدودیت‌های سامانه را ارضا کند و ثانیا باعث کاهش تعداد قبضه‌ها در هنگام تخصیص وظایف باشد. در پایان، الگوریتم زمان‌بندی ارائه‌شده روی یک پهباد نمونه و هم‌چنین سامانه‌هایی با وظایف تصادفی آزمایش می‌گردد و نرخ زمان‌بندپذیری و تعداد قبضه‌ها مقایسه می‌گردند. نتایج بدست آمده نشان می‌دهد که الگوریتم ارائه شده بدون آن‌که تغییر قابل توجهی در کاهش نرخ زمان‌بندپذیری داشته باشد تعداد قبضه‌ها را تا حدود ۹۶ درصد کاهش می‌دهد.

کلیدواژه‌ها: سامانه بی‌درنگ سامانه بحرانی-مختلط چندهسته‌ای زمان‌بندی وظایف وابسته کاهش قبضه‌ها

۱- مقدمه

پیشرفت تکنولوژی و نیاز صنعت به ویژه سامانه‌های نهفته به انجام وظایف بیشتر و محاسبات پیچیده‌تر، که در عین حال اجرای وظایف در آن‌ها با حداقل تاخیر و در زمان مقرر صورت گیرد، منجر به پررنگ‌تر شدن نقش سامانه‌های بی‌درنگ شده است. ویژگی اصلی این سامانه‌ها این است که وظایف به صورت پیش‌بینی‌پذیر و در مهلت معین خود اجرا می‌شوند.

بسیاری از سامانه‌های بی‌درنگ نهفته از بخش‌های مختلف تشکیل شده‌اند که این بخش‌ها بحرانیت مختلفی دارند و بر روی سخت‌افزار مجزایی اجرا می‌شوند. امروزه به دلایل مختلف تمایل به یکپارچه‌سازی این سامانه‌ها زیاد شده است و مفهوم سامانه‌های بحرانی-مختلط[1] را به وجود آورده است [۱]. به طوری که در یک پلتفرم سخت‌افزاری مشترک بخش‌های مختلف سامانه با بحرانیت‌های مختلف تجمیع شده‌اند. بحرانیت به مقدار تضمین در برابر خرابی که باید به یک بخش از سامانه داده شود گفته می‌شود. وظایف دارای سطح بحرانیت بالاتر به تضمین بیشتری برای اجرای درست خود نیاز دارند[۲]. سامانه‌های بحرانی-مختلط کاربردهای کاربردهای گسترده‌ای دارند. یکی از کاربردهای سامانه‌های بی‌درنگ بحرانی-مختلط در پهپادهای نظامی و ابزارهای پلیس هوشمند است. این پهپادها وظایف مختلفی از جمله مراقبت هوایی و شناسایی، رصد اهداف و زاویه حرکت و ارتباطات آن، کنترل ترافیک، تشخیص تخلف الکترونیک، کنترل مرزها و اطراف آن جهت مبارزه با قاچاقچیان و عکسبرداری برای بررسی های جغرافیایی انجام می‌دهند که ممکن است برخی از این وظایف دارای سطح بحرانیت بالاتری باشند و نیاز به تضمین بیشتری برای اجرای درست خود داشته باشند.

در سامانه‌های بحرانی-مختلط معمولا دو سطح بحرانیت در نظر گرفته می‌شود. در سامانه‌هایی با دو سطح بحرانیت، بدترین زمان اجرا (WCET[2]) برای وظایف با سطح بحرانیت بالاتر هم توسط صادرکننده گواهی[3] و هم توسط طراح سامانه محاسبه و ارزیابی می‌شود در حالی که برای وظایف با سطح بحرانیت کمتر تنها توسط طراح سامانه محاسبه می‌گردد. صادرکننده گواهی شرایط محافظه کارانه‌تری را برای ارزیابی WCET مدنظر قرار می‌دهد به همین دلیل اغلب مقدار WCET ارزیابی شده توسط صادرکننده گواهی از زمان اجرای محاسبه شده توسط طراح سامانه بیشتر است. در این سامانه‌ها هنگامی که یک وظیفه نتواند در سهمیه زمانی با بحرانیت کمتر خود (یعنی WCET ارزیابی شده توسط طراح سامانه) به صورت کامل اجرا شود سامانه به حالت با بحرانیت بیشتر تغییر حالت می‌دهد و وظایف بحرانی در سهمیه زمانی با بحرانیت بیشتر خود (یعنی WCET ارزیابی شده توسط صادرکننده گواهی) اجرا خواهند شد. در واقع در این حالت سامانه از یک حالت خوش‌بینانه به یک حالت بدبینانه و محافظه‌کارانه تغییر حالت می‌دهد.

دو چالش مهم که این سامانه‌ها با آن درگیر هستند شامل تضمین درستی و کارآمدی آنهاست [۳]. درستی در این سامانه‌ها به این معناست که وظایف با توجه به سطح بحرانیت خود، ضمانت کافی برای اجرا در مهلت معین خود داشته باشند (هرچقدر سطح بحرانیت وظیفه بالاتر باشد نیاز وظیفه برای رعایت مهلت خود بیشتر می‌شود). کارآمدی در این سامانه‌ها بدین معناست که از منابع سخت‌افزاری موجود به صورت کارا در راستای برقراری شرط درستی سامانه استفاده شود. تجمیع این دو مفهوم منجر به ارائه یک زمان‌بندی صحیح برای مجموعه وظایف در این سامانه‌ها می‌گردد. زمان‌بندی وظایف تعیین می‌کند که وظایف با چه اولویتی، در چه زمانی و در پردازنده‌های چند هسته‌ای، در کدام هسته اجرا شوند.

به منظور زمان‌بندی وظایف در سامانه های بحرانی-مختلط بسته به این‌که سامانه دارای معماری تک‌هسته‌ای یا چندهسته‌ای باشد از رویکردهای مختلفی استفاده می‌شود. استفاده از معماری چندهسته‌ای این فرصت را ایجاد می‌کند که وظایف بتوانند به صورت موازی اجرا شوند و در نتیجه بهره‌وری سامانه بالا برود. البته در کنار این مزیت، چالش افزایش پیچیدگی را هم در زمان‌بند خواهیم داشت چرا که در هنگام زمان‌بندی باید در مورد چگونگی تخصیص وظایف به هسته‌های مختلف نیز تصمیم‌گیری شود.

زمان‌بندی وظایف در سامانه‌های بحرانی-مختلط با معماری چندهسته‌ای در بسیاری از پژوهش‌ها بررسی شده است که در بسیاری از آن‌ها ارائه راهکار را با فرض مستقل بودن وظایف انجام داده‌اند. با این حال در واقعیت با بسیاری از سامانه‌ها روبرو هستیم که وابستگی داده بین وظایف مختلف وجود دارد به طوری که برخی از وظایف، وابسته به وظایف پیش‌نیاز هستند و تا زمانی که پیش نیازهای آن‌ها اجرایشان کامل نشود قادر به اجرا نخواهند بود. به طور معمول برای مدلسازی وظایف دارای وابستگی از گراف جهت‌دار بدون دور (DAG[4])

[1] Mixed-Criticality Systems
[2] Worst Case Execution Time
[3] Certification Authority
[4] Directed Acyclic Graph

استفاده می‌شود که هر گره در آن نشان‌دهنده یک وظیفه است و یال‌های جهت‌دار نشان‌دهنده جهت وابستگی وظایف هستند. برای آن که وظایف در هر گراف به صورت متناوب تکرار شوند از Multi-Periodic DAG استفاده می‌شود. در این پژوهش برای زمان‌بندی وظایف دارای وابستگی و متناوب در سامانه‌های بحرانی-مختلط چندهسته‌ای راهکاری ارائه می‌شود.

از آن جایی که بین وظایف وابستگی داده‌ای وجود دارد زمان‌بندی‌ای مطلوب است که هنگام تخصیص وظایف به هسته‌ها، هزینه ارتباط[1] بین وظایف وابسته را در نظر بگیرد. به عنوان مثال زمانی که دو وظیفه وابسته روی یک هسته اجرا شوند هزینه ارتباط نسبت به حالتی که در هسته‌های مختلف اجرا شوند کمتر است. در پژوهش‌های انجام شده در این حوزه به این هزینه کمتر توجه کرده‌اند.

مساله دیگری که در این پژوهش حائز اهمیت است کاهش تعداد قبضه[2]های بین وظایف است به طوری که یک وظیفه تا جایی که امکان دارد به صورت پیوسته روی یک هسته اجرا شود و بین اجرای آن گسستگی ایجاد نشود. وجود قبضه باعث می‌شود تا هسته‌ها در هر لحظه به وظیفه با بالاترین اولویت تخصیص پیدا کنند ولی با این حال قبضه‌های آزادانه و بدون محدودیت، موجب تحمیل هزینه بالایی به سامانه می‌شود. هنگامی که یک وظیفه در حال اجرا به نفع یک وظیفه دیگر متوقف می‌شود فرایندی طی می‌شود که خود زمان‌بر است و این سربار زمانی می‌بایست در الگوریتم زمان‌بندی لحاظ شود[۴]. در این فرایند ابتدا وظیفه کنونی متوقف شده و در صف وظایف آماده اجرا برای آینده وارد می‌شود، سپس وظیفه با اولویت بالاتر اجرا را به دست می‌گیرد. بنابراین کاهش تعداد قبضه‌ها به منظور داشتن یک الگوریتم زمان‌بندی مناسب حائز اهمیت است. علی رغم اهمیت بالای این موضوع در کاهش سربار سامانه، پژوهش‌های انجام شده در حوزه زمان‌بندی وظایف وابسته در سامانه‌های بحرانی-مختلط به این مساله کمتر توجه داشته‌اند و راهکارهای ارائه شده توسط آن‌ها شامل قبضه‌های آزادانه و بدون محدودیت است.

به طور خلاصه آنچه که در این پژوهش انجام خواهد شد شامل موارد زیر است:

- ارائه یک الگوریتم زمان‌بندی برای مجموعه وظایف وابسته و متناوب در یک سامانه بحرانی-مختلط با دو سطح بحرانیت روی یک معماری چندهسته‌ای به طوری که اولا زمان‌بندی ارائه شده صحیح باشد و ثانیا در جهت کاهش تعداد قبضه‌ها باشد.
- اعمال هزینه قبضه و ارتباط بین وظایف هنگام زمان‌بندی وظایف.
- آزمایش الگوریتم زمان‌بندی ارائه شده بر روی نرم‌افزار یک پهپاد نمونه و سامانه‌هایی با وظایف تصادفی و مقایسه نرخ زمان‌بندپذیری و تعداد قبضه‌ها.

در ادامه این مقاله، در بخش ۲ کارهای گذشته در این حوزه بررسی می‌شوند. در بخش ۳ مساله زمان‌بندی فرموله و حل می‌شود. در بخش ۴ برای یک پهپاد نمونه مساله زمان‌بندی بررسی می‌شود. در بخش ۵ زمان‌بندی روی مجموعه سامانه‌هایی با وظایف تصادفی آزمایش و تحلیل می‌گردد و در بخش ۶ نتیجه مقاله ارائه می‌شود.

۲- مرور کارهای گذشته

در این بخش، پژوهش‌های مرتبط با موضوع خود یعنی زمان‌بندی وظایف وابسته در سامانه‌های بحرانی-مختلط مورد بررسی قرار می‌گیرد. در این پژوهش‌ها، برای نمایش وظایف وابسته از DAG استفاده شده است که مجموعه وظایف سامانه به صورت گره‌هایی در یک گراف بدون دور نمایش داده می‌شوند و یال‌های جهت‌دار در این گراف نشان‌دهنده جهت وابستگی بین وظایف وابسته است. وقتی یک وظیفه به وظیفه دیگری وابسته است به این معنی است که تا زمانی که وظیفه پیش‌نیاز اجرایش تکمیل نشده است وظیفه وابسته نمی‌تواند اجرای خود را شروع کند. با توجه به تعریف سامانه‌های بحرانی-مختلط با ۲ سطح بحرانیت، هر گره در این گراف، نشان‌دهنده یک وظیفه با سطح بحرانیت بالا یا با سطح بحرانیت پایین خواهد بود.

در [۵] و [۶] زمان‌بندی وظایف وابسته در سامانه‌های بحرانی-مختلط در یک معماری تک پردازنده‌ای مورد بررسی قرار گرفته است. یک ترتیب توپولوژیک از گره‌ها ساخته می‌شود و گره‌هایی که پیش نیازی ندارند یا پیش نیاز آنها قبلا آماده شده‌اند به ترتیب وارد یک لیست از گره‌های آماده می‌شوند. در این ترتیب توپولوژیک همیشه وظایف با سطح بحرانیت بالا اولویت بالاتری نسبت به وظایف با سطح بحرانیت پایین دارند. الگوریتم ارائه شده در این مقاله‌ها بهینه است ولی فقط برای حالت تک پردازنده‌ای قابل استفاده است که امکان موازیسازی وظایف را سلب می‌کند.

در [۷] زمان‌بندی وظایف وابسته در سامانه‌های بحرانی-مختلط در یک معماری چند پردازنده‌ای مورد بررسی قرار گرفته است و فرض شده است که کل وظایف DAG دارای مهلت یکسان و مشترک هستند. ابتدا جدول زمان‌بندی وظایف در حالت با بحرانیت بالای سامانه را با اعمال روش LS[3] [۸] به صورت غیر قبضه‌ای روی گره‌های

[1] Communication Cost
[2] preemption
[3] List Scheduling

با سطح بحرانیت بالا می‌سازد و ترتیب آن‌ها را ذخیره می‌کند. سپس جدول زمان‌بندی در حالت با بحرانیت پایین سامانه را با اعمال LS به صورت قبضه‌ای و با استفاده از ترتیب به دست آمده، روی تمامی گره‌ها می‌سازد. روش استفاده شده در این مقاله بهینه نیست. هم چنین اولویت تام وظایف با سطح بحرانیت بالا نسبت به وظایف با سطح بحرانیت پایین باعث می‌شود که در حالت سامانه با بحرانیت پایین امکان زمان‌بندپذیری وظایف با سطح بحرانیت پایین کاهش یابد.

در [۹] با تعریف مفهومی با عنوان آخرین لحظه فعال‌سازی ایمن (LSAI[1]) درصدد برآمده است تا اولویت تام دادن وظایف با سطح بحرانیت بالا نسبت به وظایف با سطح بحرانیت پایین را حذف کند و مشکلی را که در زمان‌بندی قبلی موجب ایجاد محدودیت سختگیرانه در زمان‌بندی در حالت با بحرانیت پایین می‌شد، برطرف سازد. LSAI ها در واقع زمان شروع وظایف با سطح بحرانیت بالا در جدول زمان‌بندی با حالت بحرانیت بالای سامانه هستند و تعیین‌کننده لحظاتی هستند که می‌بایست به وظایف با سطح بحرانیت بالا اولویت بالاتری نسبت به وظایف با سطح بحرانیت پایین داده شود. بنابراین به عنوان یک مهلت مجازی در هنگام زمان‌بندی در حالت با بحرانیت پایین سامانه در نظر گرفته می‌شوند. در این مقاله، برای مجموعه‌ای از وظایف که در قالب DAG های چندگانه با مهلت‌ها یا دوره‌های تناوب متفاوت هستند راهکاری ارائه نشده است.

در [۱۰] زمان‌بندی DAG های چندگانه با دوره‌های تناوب متفاوت در یک سامانه بحرانی-مختلط با معماری چندپردازنده‌ای مورد بررسی قرار گرفته است. در این مقاله فرض شده است که هر DAG، یک وظیفه با دوره تناوب خاص خود است و گره‌های آن DAG، در واقع job های آن وظیفه هستند که بین آن‌ها وابستگی وجود دارد. همچنین سطح بحرانیت به جای این که به وظایف اختصاص داده شود به job ها اختصاص داده شده است. برای تخصیص پردازنده به وظایف از رویکرد فدرال[2] استفاده شده است که در آن وظایف بر اساس بهره‌برداری[3] به وظایف سبک و سنگین تقسیم می‌شوند. برای یک وظیفه سبک، همه job های آن روی یک پردازنده و برای وظایف سنگین روی چندین پردازنده اجرا خواهند شد. در این مقاله همانند [۷]، به job های با سطح بحرانیت بالا همواره اولویت بالاتری اختصاص می‌دهد که موجب کاهش امکان زمان‌بندپذیری در حالت با بحرانیت پایین سامانه می‌شود. روش فدرال از نظر استفاده از منابع ، کارا نیست چرا که پردازنده‌ها به صورت اختصاصی به DAG ها اختصاص می‌یابند و به منظور داشتن یک زمان‌بندی موفق از تعداد پردازنده‌های بالایی استفاده خواهد کرد. هم چنین در وظایف سبک امکان موازی‌سازی job ها را سلب می‌کند.

در [۱۱] از روشی تحت عنوان شبه فدرال استفاده می‌شود. هدف اصلی که در این مقاله دنبال می‌شود این است که از هدررفت پردازنده در روش فدرال جلوگیری کند. در این مقاله هم سطوح بحرانیت برای job های یک وظیفه تعریف شده است و وابستگی بین job های یک وظیفه در قالب یک DAG نمایش داده می‌شود. این روش هم همانند روش فدرال، به دلیل اولویت تام دادن به job های با بحرانیت بالا ، امکان زمان‌بندپذیری وظایف با سطح بحرانیت پایین را کاهش می‌دهد.

در [۱۲] و [۱۳] نیز از روش فدرال برای تخصیص هسته به DAG های چندگانه با دوره‌های تناوب متفاوت استفاده می‌شود. در مدل ارائه شده توسط آن‌ها، هر DAG معادل یک وظیفه با سطح بحرانیت بالا یا سطح بحرانیت پایین در نظر گرفته می‌شود و گره‌های موجود در DAG، job های آن وظیفه هستند که بین آن‌ها وابستگی وجود دارد. بنابراین یک DAG و همه گره‌های موجود در آن همگی از یک سطح بحرانیت خواهند بود و در واقع بین وظایف با سطوح بحرانیت مختلف هیچ وابستگی‌ای وجود ندارد.

در [۱۴] از روش سراسری برای زمان‌بندی وظایف وابسته در سامانه‌های بحرانی-مختلط با معماری چندهسته‌ای استفاده شده است. در روش‌های سراسری وظایف برای اجرا در هسته‌های مختلف هیچ محدودیتی ندارند. در مدل ارائه شده توسط این مقاله هر سامانه بحرانی-مختلط توسط چندین DAG از وظایف نشان داده می‌شود که هر DAG برای خود یک دوره تناوب جداگانه دارد. هم‌چنین وابستگی وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین مجاز نمی‌باشد. جداول زمان‌بندی در حالت سامانه با بحرانیت بالا و بحرانیت پایین به صورت استاتیک ساخته می‌شود. ابتدا جدول زمان‌بندی در حالت با بحرانیت بالای سامانه با روش G-LLF[4] و به صورت ALAP[5] ساخته می‌شود. سپس جدول زمان‌بندی در حالت با بحرانیت پایین سامانه را با روش G-LLF و به صورت ASAP[6] می‌سازد. هم چنین برای این که انتقال حالت از با بحرانیت کمتر به حالت با بحرانیت بالاتر امن باشد و اطمینان حاصل شود که وظایف مهلت‌های خود را رعایت می‌کنند یک شرط انتقال امن تعریف شده است. روش ارائه شده در این مقاله از کمینه تعداد هسته استفاده می‌کند و امکان زمان‌بندپذیری نسبت به روش‌های قبلی در آن بالاتر است. در این

[1] Latest Safe Activation Instant
[2] federated
[3] utilization
[4] Global-Least Laxity First
[5] As Late As Possible
[6] As Soon As Possible

روش هیچ کنترلی روی تعداد قبضه‌های وظایف نیست و بسیاری از قبضه‌ها غیرضروری هستند. هم‌چنین هزینه قبضه و هزینه ارتباط بین وظایف وابسته نیز در زمان‌بندی لحاظ نشده است.

در [۱۵] روش ارائه شده در [۱۴] برای یک سامانه بحرانی-مختلط با تعداد سطوح بحرانی دلخواه تعمیم داده شده است. برای ساخت جداول زمان‌بندی از سه روش G-LLF، G-EDF و G-EDZL[1] استفاده می‌کند. از بالاترین حالت بحرانی شروع می‌کند و جدول زمان‌بندی در آن حالت بحرانی را با یکی از روش‌های مذکور و به صورت ALAP می‌سازد. ساخت جدول زمان‌بندی به این روش را تا یک حالت قبل از پایین‌ترین حالت بحرانی ادامه می‌دهد. برای پایین‌ترین حالت بحرانی جدول زمان‌بندی را با یکی از همان سه روش و به صورت ASAP می‌سازد. در این مقاله برخلاف سایر پژوهش‌های انجام شده در این حوزه، وابستگی وظایف با سطح بحرانیت بالاتر به وظایف با سطح بحرانیت پایین‌تر مجاز شمرده است ولی در هنگام زمان‌بندی فرض کرده است که چنین وظایفی بدون آن که نیاز به تکمیل اجرای پیش‌نیاز های خود داشته باشند می‌توانند اجرا شوند و وابستگی را نقض می‌کند. روش ارائه شده در این مقاله نیز با توجه به اینکه از رویکرد زمان‌بندی سراسری استفاده می‌کند قبضه‌های زیادی تولید می‌کند.

۳- حل مساله زمان‌بندی

به منظور حل مساله زمان‌بندی، ابتدا مدل سامانه به صورت دقیق مشخص می‌شود و سپس شرایط صحت زمان‌بندی برای سامانه مورد نظر بیان می‌شود. در نهایت الگوریتم زمان‌بندی شرح داده می‌شود.

۳-۱- مدل‌سازی سامانه

در این پژوهش یک سامانه بحرانی-مختلط (MCS) به نام S با دو حالت بحرانیت پایین (Low) و بالا (High) در نظر گرفته می‌شود. سامانه از حالت با بحرانیت پایین شروع به کار می‌کند و همه وظایف از سهمیه زمانی با بحرانیت پایین خود برای اجرا استفاده می‌کنند. هر هنگام که یک وظیفه نتواند در این سهمیه زمانی اجرایش را کامل کند، یک TFE[2] (رویداد شکست زمان‌بندی) رخ می‌دهد و سامانه تغییر حالت داده و به حالت با بحرانیت بالا می‌رود و از این به بعد وظایف از سهمیه زمانی با بحرانیت بالای خود برای اجرا استفاده می‌کنند. در هنگام تغییر حالت سامانه، برای وظایف با سطح بحرانیت Low از رویکرد دور انداختن استفاده می‌شود بنابراین وظایف با سطح بحرانیت Low در حالت با بحرانیت بالای سامانه متوقف می‌شوند [۱۶].

سامانه S به صورت $S = (\Pi, \Gamma)$ تعریف می‌شود. Π بیان‌گر معماری سامانه است و شامل m هسته همگن و یکسان است. Γ نیز بیان‌گر مجموعه‌ای از DAG های بحرانی-مختلط (MC-DAG) است که هر کدام از آن‌ها با G_i نمایش داده می‌شود. به ازای هر G_i که عضو مجموعه Γ است سه مشخصه خواهیم داشت:

- V_i مجموعه‌ای از گره‌های موجود در G_i است. هر گره یک وظیفه بحرانی-مختلط (MC-Task) است که با τ_j نشان داده می‌شود. زمان رهاسازی تمامی وظایف موجود در G_i یکسان و برابر صفر است.
- E_iمجموعه‌ای از یال‌های موجود در G_i است و زیرمجموعه‌ای از $V_i \times V_i$ است. هر یال عضو E_iیک وابستگی بین دو وظیفه موجود در G_i را نشان می‌دهد که جهت یال بیان‌گر جهت وابستگی است و به صورت دوتایی (τ_a, τ_b) نمایش داده می‌شود. به τ_a وظیفه پیشین و به τ_b وظیفه پسین گفته می‌شود. با استفاده از مجموعه E_i و به ازای هر وظیفه τ_j، مجموعه $pred(\tau_j)$ (وظایف پیشین یا وظایف پیش‌نیاز τ_j) و مجموعه $succ(\tau_j)$ (وظایف پسین τ_j) استخراج می‌شوند. هر وظیفه τ_j، تا زمانی که تمام وظایف عضو $pred(\tau_j)$ اجرایشان کامل نشود نمی‌تواند شروع به اجرا کند. هم‌چنین با توجه به استفاده از رویکرد دور انداختن، وظیفه دارای سطح بحرانیت $High$ نمی‌تواند به وظیفه با سطح بحرانیت Low وابسته باشد.
- T_i یک عدد طبیعی و نشان‌دهنده دوره تناوب G_i است. وظایف موجود در G_i، در هر T_i واحد زمانی، بازتولید می‌شوند. چون دوره تناوب مربوط به هر MC-DAG می‌تواند مقداری متفاوت از دیگری باشد بنابراین سامانه S، Multi-Periodic است. یک سامانه Multi-Periodic که زمان رهاسازی تمام وظایف آن یکسان است هنگامی زمان‌بندپذیر است که در یک Hyper-Period زمان‌بندپذیر باشد[۸]. بنابراین برای این که سامانه S زمان‌بندپذیر باشد باید در یک Hyper-Period زمان‌بندپذیر بودن آن آزمایش شود. Hyper-Period یک سامانه برابر با کوچکترین مضرب مشترک بین دوره تناوب وظایف آن سامانه است، بنابراین مقدار Hyper-Period برای S از رابطه $HP(S) = lcm_{G_i \in \Gamma} T_i$ حساب می‌شود.

به ازای هر MC-Task به نام τ_j که عضو مجموعه V_i است پنج مشخصه خواهیم داشت:

[1] Global-Earliest Deadline Zero Laxity
[2] Timing Failure Event

- L_j سطح بحرانیت وظیفه τ_j است و Low یا $High$ است.
- $C_j(Low)$ برابر با WCET وظیفه τ_j در حالت با بحرانیت پایین سامانه است که یک عدد نامنفی و نشان‌دهنده سهمیه زمانی یا زمان اجرای وظیفه τ_j در حالت سامانه با بحرانیت پایین است.
- $C_j(High)$ برابر با WCET وظیفه τ_j در حالت با بحرانیت بالای سامانه است که یک عدد نامنفی و نشان‌دهنده سهمیه زمانی یا زمان اجرای وظیفه τ_j در حالت سامانه با بحرانیت بالا است و برای وظایف با سطح بحرانیت$High$ همواره مقدار آن بزرگتر یا مساوی با مقدار $C_j(Low)$است. هم‌چنین با توجه به رویکرد دور انداختن، برای وظیفه τ_j با سطح بحرانیت Low آنگاه $C_j(High) = 0$ خواهد بود.
- T_j یک عدد طبیعی و نشان‌دهنده دوره تناوب وظیفه τ_j است که مقدار آن برابر با دوره تناوب گرافی است که شامل وظیفه τ_j است ($T_j = T_i$). وظیفه τ_j در هر T_j واحد زمانی، یکبار بازتولید می‌شود که هر بازتولید آن را یک job می‌گویند. k امین بازتولید یا job وظیفه τ_j را با $J_{j,k}$ نشان می‌دهند و تمامی مشخصه‌های وظیفه τ_j را به ارث می‌برد. در واقع job ها نمودهای واقعی یک وظیفه متناوب در مساله زمان‌بندی هستند و در نهایت این job ها هستند که زمان‌بندی می‌شوند. زمان رهاسازی[1] $J_{j,k}$ از طریق رابطه $r_{j,k} = (k-1) \times T_j$ و مهلت نسبی آن از رابطه $d_{j,k} = k \times T_j$ محاسبه می‌شود. اجرای $J_{j,k}$ باید بین مقدار $r_{j,k}$ و $d_{j,k}$ باشد تا زمان‌بندی مجاز باشد.
- D_j نشان‌دهنده مهلت مطلق وظیفه τ_j است. مهلت وظیفه به صورت implicit-deadline در نظر گرفته می‌شود و مقدار آن با دوره تناوب آن وظیفه یکسان خواهد بود یعنی $D_j = T_j$.

مقدار بهره‌برداری برای هر گراف G_iدر حالت سامانه با بحرانیت θ از طریق رابطه $U_\theta(G_i) = \sum_{\tau_j \in G_i} C_j(\theta)/T_i$ به دست می‌آید و مقدار بهره‌برداری کل سامانه در حالت با بحرانیت θ از طریق رابطه $U_\theta(S) = \sum_{G_i \in \Gamma} U_\theta(G_i)$ محاسبه می‌شود. طبق [۸] به منظور این که یک سامانه زمان‌بندپذیر باشد باید تعداد هسته‌های سامانه از بهره‌برداری سامانه در هر حالت بحرانی بیشتر یا مساوی باشد.

۳-۲- شرایط صحت زمان‌بندی

طبق [۱۰] و [۱۴] برای آن که یک زمان‌بندی ارائه شده برای سامانه S صحیح باشد باید این شرایط رعایت شوند:

صحت زمان‌بندی در حالت سامانه با بحرانیت پایین :
هنگامی‌که S در حالت با بحرانیت پایین قرار دارد (حالتی که زمان اجرای هیچ‌کدام از وظایف سامانه از مقدار $C(Low)$آن‌ها تجاوز نمی‌کند)، باید تمامی وظایف سامانه تا قبل از رسیدن مهلتشان اجرایشان را تکمیل کرده باشند. به بیان دقیق‌تر، برای هر وظیفه τ_j موجود در S که در k امین فعالسازی یا بازتولید خود است، باید مقدار زمان تخصیص یافته به τ_j در فاصله زمانی بین رهاسازی و مهلت آن در حالت سامانه با بحرانیت پایین از مقدار $C_j(Low)$ کمتر یا برابر با این مقدار باشد.

صحت زمان‌بندی در حالت سامانه با بحرانیت بالا :
هنگامی‌که S در حالت با بحرانیت بالا قرار دارد (حالتی که زمان اجرای هیچ‌کدام از وظایف سامانه از مقدار $C(High)$آن‌ها تجاوز نمی‌کند) ، باید تمامی وظایف سامانه که دارای سطح بحرانیت $High$ هستند تا قبل از رسیدن مهلتشان اجرایشان را تکمیل کرده باشند. به بیان دقیق‌تر، برای هر وظیفه τ_j موجود در S که در k امین فعالسازی یا بازتولید خود است و سطح بحرانیت آن $High$ است، باید مقدار زمان تخصیص یافته به τ_j در فاصله زمانی بین رهاسازی و مهلت آن در حالت سامانه با بحرانیت بالا از مقدار $C_j(High)$ کمتر یا برابر با این مقدار باشد.

شرط انتقال امن : این شرط در واقع مرتبط با صحت زمان‌بندی در حالت با بحرانیت بالای سامانه است و به طور ویژه‌تر به تاثیر تغییر حالت سامانه روی صحت زمان‌بندی وظایف با سطح بحرانیت $High$ می‌پردازد. طبق این شرط، هنگامی که سامانه از حالت با بحرانیت پایین به حالت با بحرانیت بالا تغییر حالت می‌دهد باید تضمین شود این تغییر حالت سامانه امن است یعنی مدت زمان کافی برای تکمیل اجرای وظایف با سطح بحرانیت $High$ وجود داشته باشد. طبق اثباتی که در [۱۴] انجام شده است به ازای هر وظیفه τ_j با سطح بحرانیت $High$ موجود در S که در k امین بازتولید خود است و در هر زمان t باید رابطه (۱) برقرار باشد تا انتقال امن سامانه تضمین شود.

$$\forall \tau_j , et_{\tau_j}^{Low}(r_{j,k}, t) < C_j(Low) : \quad et_{\tau_j}^{Low}(r_{j,k}, t) \geq et_{\tau_j}^{High}(r_{j,k}, t) \qquad (۱)$$

در این رابطه t بیان‌گر زمان وقوع TFE در سامانه است و از آنجا که قبل از اجرا زمان وقوع TFE مشخص نیست باید برای بازه زمانی $r_{j,k} \leq t \leq d_{j,k}$ صحت این شرط بررسی شود. تابع $et_\tau^\theta(t_1, t_2)$ برابر با زمان اجرای سپری شده از وظیفه τ در فاصله زمانی بین t_1 و t_2 در حالت سامانه با بحرانیت θ است. طبق رابطه (۱)، در وظایف با سطح

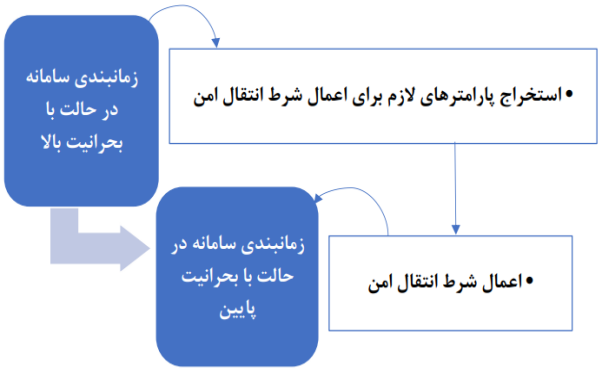
[1] Release time

بحرانیت $High$ که تا قبل از t اجرایشان در حالت بـا بحرانیـت پـایین سامانه تمام نشده است می‌بایست مقدار زمان تخصیص داده شـده بـه آن وظیفه تا زمان t در حالت سامانه با بحرانیت پایین از مقـدار زمـان تخصیص داده شده به آن وظیفه تا زمان t در حالت سامانه با بحرانیت بالا بزرگتر یا مساوی باشد.

۳-۳- زمان‌بندی وظایف

برای زمان‌بندی وظایف در سامانه S از یک الگوریتم استاتیک اسـتفاده خواهیم کرد به این معنا که قبل از زمان اجـرا و بـا توجـه بـه مـدل و مشخصات سامانه S، تخصیص زمـان و هسـته هـر کـدام از وظـایف در حالت سامانه با بحرانیت پایین و حالت سامانه بـا بحرانیـت بـالا انجـام می‌شود و نتایج آن‌ها در $table^{Low}$ (جدول زمـان‌بنـدی در حالـت بـا بحرانیت پایین سامانه S) و $table^{High}$ (جدول زمان‌بندی در حالت با بحرانیت بالای سامانه S) ذخیره می‌شود. استاتیک بودن الگوریتم ایـن امکان را می‌دهد تا شرط انتقال امن که در بخش قبل به آن اشاره شـد به راحتی محاسبه و بررسی شود. از جملـه روش‌هـای اسـتاتیک بـرای زمان‌بندی می‌توان به LLF و EDF اشاره کرد [۱۷].

زمان‌بندی ارائه شده در این پژوهش دو قدم اساسی دارد. در قدم اول بایـد زمـان‌بنـدی در حالـت بـا بحرانیـت پـایین سـامانه انجـام و $table^{Low}$ تشکیل شود. اگر زمان‌بندی انجـام شـده مطـابق بـا شـرط صحت زمان‌بندی در حالت سامانه با بحرانیت پایین درست بـود، آنگـاه در قدم دوم زمان‌بندی در حالت با بحرانیت بالای سامانه با اعمال شرط انتقال امن انجام شده و $table^{High}$ تشکیل می‌شود. اگـر زمـان‌بنـدی انجام شده مطابق بـا شـرط صـحت زمـان‌بنـدی در حالـت سـامانه بـا بحرانیت بالا هم درست بود کل زمان‌بندی انجام شده بـرای سـامانه S درست ارزیابی می‌شود چرا که هر سه شرط صحت زمان‌بندی سامانه را رعایت کرده است. در شکل (۱) مراحل کلی زمان‌بنـدی سـامانه آمـده است.



شکل (۱): مراحل کلی زمان‌بندی سامانه

نحوه استفاده سامانه از این زمان‌بندی در زمان اجرا به این صورت است که تا قبـل از وقـوع TFE، از $table^{Low}$ بـرای تخصـیص وظـایف استفاده می‌شـود و بعـد از وقـوع TFE، از $table^{High}$ بـرای تخصـیص وظایف استفاده خواهد شد.

در الگوریتم ارائه شده در [۱۴] که بیشـترین مشـابهت را از نظـر تعریف مساله با این پژوهش دارد زمان‌بندی به صورت قبضـه‌ای انجـام شده است و هیچ کنترلی روی تعداد قبضه‌های تولیدی ندارد. در حالی که بسیاری از این قبضه‌ها غیرضروری هستند و سامانه بدون آن‌ها هـم می‌تواند زمان‌بندپذیر باشد. هم چنین هزینه قبضه و ارتباط در هنگـام تخصیص وظایف در نظر گرفته نشده است و زمان‌بندی در یک سـامانه ایده آل مورد بررسی قـرار گرفتـه اسـت کـه ممکـن اسـت در هنگـام استفاده برای سامانه‌های واقعی کارا نباشـد. در واقـع در نظـر نگـرفتن هزینه قبضه و ارتباط باعث می‌شود زمان‌بنـدی انجـام شـده در حالـت تئوری، درست به نظر برسد در حالی کـه در کـاربرد ممکـن اسـت آن زمان‌بندی به علت در نظر نگرفتن این هزینه‌ها شکست بخورد. در ایـن پژوهش در راستای کاهش تعـداد قبضـه‌هـا، زمـان‌بنـدی بـه صـورت قبضه‌ای محدود ارائه می‌شـود. هـم‌چنـین هزینـه ارتبـاط و قبضـه در الگوریتم زمان‌بندی لحاظ می‌شود.

هزینه قبضه : در سـامانه S، اگـر از زمـان‌بنـدی قبضـه‌ای بـرای تخصیص وظایف استفاده شود آن‌گاه وظیفه می‌توانـد در بـین اجـرای خود، متوقف شود و ادامه اجرای خود را بر روی یک هسته دیگر یا یک زمان دیگر انجام دهد. وجـود گسسـتگی بـین اجـرای وظیفـه موجـب تحمیل یک سربار زمانی می‌شود که بـه آن هزینـه قبضـه مـی‌گوینـد. هزینـــــه قبضـــــه هـــــر وظیفـــــه τ_j از رابطـــــه $preemption_{cost}(\tau_j) = \lceil PF \times execution(\tau_j) \rceil$ به دست می‌آیـد [۱۸]. در این رابطه [1] PF عددی ثابت در کل سامانه و در بازه $[0,0.5]$ است و $execution(\tau_j)$ برابر با زمان اجرای وظیفه است. (اگر سـامانه در حالت با بحرانیت پایین باشد زمان اجرای وظیفه برابر با $C_j(Low)$ و اگر سامانه در حالت با بحرانیت بالا باشد زمان اجرای وظیفـه برابـر بـا $C_j(High)$ خواهد بود.)

هزینه ارتباط : در سامانه S، اگر یال (τ_a, τ_b) در سامانه وجـود τ_a به برخی از داده‌های خروجی وظیفـه τ_bداشته باشد آن‌گاه وظیفه τ_b به اتمام نرسیده τ_a تا زمانی که اجرای وظیفه τ_bوابسته است و وظیفه باشد اجازه شروع فعالیت ندارد. با توجه به هسته میزبان این دو وظیفه، دو حالت ممکن است رخ دهد [۸]. اگر این دو وظیفـه بخـش ابتـدایی روی هسته مشترکی زمان‌بنـدی τ_aو بخش انتهایی وظیفه τ_bوظیفه

[1] Preemption Factor

شده باشند انتقال داده بین آن‌ها در صفر واحد زمانی صورت می‌گیرد $communication_{cost}(\tau_a,\tau_b)=0$. اگر این دو وظیفه روی هسته‌های متفاوتی زمان‌بندی شده باشند، وظیفه τ_b می‌بایست پیش از شروع به اجرای خود، مدت زمانی را صرف انتقال داده از وظیفه τ_a کند. این مدت زمان انتقال داده برابر با $communication_{cost}(\tau_a,\tau_b)=\lfloor CF\times execution(\tau_b)\rfloor$ خواهد بود. در این رابطه CF[1] عددی ثابت در کل سامانه و در بازه [0,0.5] است. $execution(\tau_b)$ هم برابر با زمان اجرای وظیفه τ_b است. حال اگر وظیفه τ به مجموعه‌ای از وظایف وابسته باشد باید به اندازه $\max_{\tau_p\in pred(\tau)} communication_{cost}(\tau_p,\tau)$ قبل از شروع به اجرای خود صرف انتقال داده کند.

به منظور پیاده‌سازی زمان‌بندی سامانه، یک الگوریتم برای محاسبه جدول زمان‌بندی در حالت سامانه با بحرانیت پایین و یک الگوریتم برای محاسبه جدول زمان‌بندی در حالت سامانه با بحرانیت بالا خواهیم داشت.

۳-۳-۱- الگوریتم محاسبه جدول زمان‌بندی در حالت با بحرانیت پایین سامانه

شبه کد این الگوریتم در بخش ضمایم نمایش داده شده است و ورودی آن سامانه S است. پیش از توضیح الگوریتم، سه لیست اصلی که در اجرای این الگوریتم نقش مهمی ایفا می‌کنند معرفی می‌شوند:

- $Ready_{list}$: لیست وظایفی که آماده اجرا هستند و در شروع الگوریتم این لیست خالی است.
- $Running_{list}$: لیست وظایفی که در حال اجرا هستند و در شروع الگوریتم این لیست خالی است.
- $Idle_{list}$: لیست هسته‌هایی که بی‌کار هستند و در شروع الگوریتم شامل تمام هسته‌های سامانه است.

در ابتدا اولین فعالسازی از همه وظایف موجود در تمام DAGهای سامانه به منظور زمان‌بندی وارد می‌شوند و مدت زمان باقی مانده از اجرای هر وظیفه یا $remaining(\tau_j)$ برابر با سهمیه زمانی در حالت با بحرانیت پایین آن‌ها قرار می‌گیرد. $Ready_{list}$ به روز رسانی می‌شود و وظایفی که هیچ وظیفه پیش نیازی ندارند (وظایف جد) وارد این لیست می‌شوند. سپس برای همه زمان‌های بین ۰ تا $HP(S)$ مراحل زیر تکرار می‌شود تا نحوه تخصیص وظایف در هر کدام از زمان‌های این بازه مشخص شوند. در هر زمان t این مراحل انجام می‌شود:

۱) مرتب سازی وظایف آماده به اجرا: وظایف موجود در $Ready_{list}$ بر اساس لختی[2] خود در زمان t به صورت صعودی مرتب سازی می‌شوند. هرچقدر لختی یک وظیفه کمتر باشد اولویت آن برای اجرا بیشتر خواهد بود. لختی برابر با فرجه زمانی هر وظیفه به منظور اجرای صحیح و تا قبل از فرارسیدن مهلت آن است. لختی در زمان t و برای هر وظیفه τ_j که در k امین فعالسازی خود است ($J_{j,k}$) از طریق رابطه (۲) محاسبه می‌شود [۱۴].

$$laxity(\tau_j,t)=d_{j,k}-t-CP(\tau_j)-remaining(\tau_j) \quad (۲)$$

در این رابطه $CP(\tau_j)$ برابر با طول مسیر بحرانی (طول بزرگترین مسیر از گره τ_j تا گره خروجی) است. تاثیر مقدار $CP(\tau_j)$ در فرمول $laxity(\tau_j,t)$ باعث می‌شود که یک نگاه رو به جلو در تعیین فرجه وظیفه τ_j داشته باشیم. در واقع در محاسبه فرجه وظیفه τ_j ، علاوه بر این که اجرای صحیح و تا قبل از مهلت وظیفه τ_j را در نظر می‌گیرد اجرای صحیح و تا قبل از فرجه وظایف پسین τ_j را نیز لحاظ می‌کند.

اگر $laxity(\tau_j,t)=0$ باشد آن‌گاه وظیفه τ_j یک وظیفه اضطراری است و باید در زمان t زمان‌بندی شود تا خودش یا وظایف پسین آن قبل از رسیدن مهلتشان، اجرایشان را به اتمام برسانند.

۲) تخصیص وظایف: بعد از آن که اولویت وظایف در $Ready_{list}$ مشخص شد، وظایف به ترتیب اولویت خود در زمان t زمان‌بندی می‌شوند. هنگام تخصیص وظایف ممکن است سه حالت رخ دهد:

- اگر هسته بی‌کار وجود داشته باشد ($Idle_{list}\neq\emptyset$) آن‌گاه وظیفه با بیشترین اولویت ($Ready_{list}[0]$) برای زمان‌بندی در لحظه t انتخاب می‌شود (τ_x). سپس هسته میزبان آن وظیفه (c_x) از بین هسته‌های بی‌کار به گونه‌ای انتخاب می‌شود که کمترین هزینه ارتباط را به آن وظیفه تحمیل کند. در واقع از بین هسته‌های بی‌کار، هسته‌ای انتخاب شود که مقدار $\max_{\tau_p\in pred(\tau_x)} communication_{cost}(\tau_p,\tau_x)$ را کمینه کند.
- اگر هسته بی‌کاری وجود نداشته باشد ($Idle_{list}=\emptyset$) و وظیفه $Ready_{list}[0]$ دارای لختی صفر باشد ($laxity(Ready_{list}[0],t)=0$)، یک وظیفه در حال اجرا در صورت اضطراری نبودن خودش، متوقف می‌شود و هسته‌ی میزبان آن را، وظیفه $Ready_{list}[0]$ تصاحب می‌کند. زیرا وظیفه $Ready_{list}[0]$ در لحظه t هیچ فرجه‌ای ندارد و در صورت اجرا نشدن در لحظه t ، مهلت آن یا وظایف پسین آن در آینده رعایت نخواهد شد. از بین وظایف در حال اجرا، وظیفه با بیشترین لختی انتخاب می‌شود (وظیفه در حال اجرا با کمترین اولویت) و در

[1] Communication Factor

[2] Laxity

صورتی که لختی آن بیشتر از صفر باشد ($laxity(\tau_y, t) > 0$) ، متوقف می‌شود و از لیست وظایف در حال اجرا خارج و به لیست وظایف آماده به اجرا منتقل می‌شود. هم‌چنین هسته میزبان وظیفه متوقف شده (τ_y) در لحظه t به حالت بی‌کار در می‌آید و این هسته (c_x) در اختیار وظیفه (τ_x) $Ready_{list}[0]$ قرار می‌گیرد. جدول زمان‌بندی هم با توجه به متوقف شدن τ_y، به روزرسانی می‌شود و تخصیص وظیفه τ_y در زمان‌های بزرگتر یا مساوی t از آن حذف می‌شود.

- اگر هسته بی‌کاری وجود نداشته باشد ($Idle_{list} = \emptyset$) و وظیفه $Ready_{list}[0]$ که در ابتدای لیست وظایف آماده به اجرا قرار دارد اضطراری نباشد ($laxity(Ready_{list}[0], t) > 0$) تخصیص در زمان t به اتمام می‌رسد و به مرحله (۳) می‌رود.

بعد از مشخص شدن τ_x و c_x ، $load(\tau_x)$ (زمان بارگذاری یا آماده سازی وظیفه) مقداردهی می‌شود. اگر τ_x وظیفه‌ای باشد که قبلا قبضه شده است ($remaining(\tau_x) < C_{\tau_x}(Low)$)، مقدار $load(\tau_x)$ برابر با هزینه قبضه آن وظیفه قرار می‌گیرد و در غیر این صورت، $load(\tau_x)$ برابر با هزینه ارتباط قرار می‌گیرد. هر وظیفه‌ی آماده اجرا، بعد از گذراندن مقدار $load$ خود می‌تواند اجرای اصلی خود را شروع کند. سپس شرط صحت زمان‌بندی (VerifyDeadline(τ_x, t)) برای آن وظیفه بررسی شود و اگر تشخیص داده شود زمان‌بندی وظیفه در زمان t موجب نقض مهلت آن وظیفه یا وظایف پسین آن می‌شود زمان‌بندی غلط اعلام شده و الگوریتم به پایان می‌رسد ($laxity(\tau_x, t) - load(\tau_x) < 0$). اگر صحت زمان‌بندی تضمین شود تخصیص در جدول زمان‌بندی انجام می‌شود و از زمان t تا $t + load(\tau_x)$ به بارگذاری آن وظیفه و از زمان $t + load(\tau_x)$ تا $t + load(\tau_x) + remaining(\tau_x)$ به اجرای اصلی وظیفه روی هسته c_xاختصاص داده می‌شود. سپس وظیفه زمان‌بندی شده از $Ready_{list}$ حذف و به $Running_{list}$ می‌رود. هم‌چنین c_xاز $Idle_{list}$ خارج می‌شود.

۳) به روز رسانی لیست وظایف در حال اجرا و آماده به اجرا: به ازای هر وظیفه موجود در $Running_{list}$ ، مقدار $remaining$ یا $load$ آن وظیفه یک واحد کم می‌شود. (در صورتی که وظیفه هنوز زمان بارگذاری خود را کامل نکرده باشد $load$ کاهش می‌یابد و در غیر این‌صورت $remaining$ کاهش می‌یابد). سپس وظایف پسینِ وظایفی که اجرای خود را تکمیل کرده‌اند در صورتی که همه پیش‌نیاز هایشان اجرایشان تکمیل شده باشد به لیست وظایف آماده به اجرا اضافه می‌شوند و وظایفی که اجرایشان تکمیل شده است از لیست وظایف در حال اجرا خارج می‌شوند.

۴) بازفعالسازی وظایف: در صورتی که یک DAG، به زمان دوره تناوب خود برسد باید مجددا فعالسازی شود و فعالسازی جدید وظایف آن (job های جدید) به منظور زمان‌بندی وارد شوند. زمان اجرای باقیمانده هر کدام از این وظایف برابر با سهمیه زمانی آن‌ها در حالت با بحرانیت پایین سامانه قرار می‌گیرد. هم چنین وظایفی که پیش‌نیازی ندارند وارد $Ready_{list}$ می‌شوند.

هنگامی که برای همه زمان‌ها تا Hyper-Period این مراحل انجام شد الگوریتم زمان‌بندی در حالت با بحرانیت پایین سامانه به اتمام می‌رسد و جدول زمان‌بندی در این حالت را به عنوان خروجی می‌دهد.

۳-۳-۲- الگوریتم محاسبه جدول زمان‌بندی در حالت با بحرانیت بالای سامانه

شبه کد این الگوریتم در بخش ضمایم نمایش داده شده است و ورودی آن سامانه S است. این الگوریتم شباهت زیادی با الگوریتم محاسبه جدول زمان‌بندی در حالت با بحرانیت پایین سامانه دارد و فقط در برخی موارد جزئی با آن تفاوت دارد که آن‌ها را بیان می‌کنیم:

- حذف وظایف با سطح بحرانیت Low: طبق رویکرد دورانداختن، در حالت با بحرانیت بالای سامانه، وظایف با سطح بحرانیت Low حضور ندارند و هنگام زمان‌بندی در این حالت باید تمام این وظایف و یال‌های مرتبط با آن‌ها از DAG های موجود در سامانه حذف شوند. هم‌چنین وظایف از سهیمه زمانی با بحرانیت بالای خود یعنی $C(High)$ استفاده می‌کنند.
- اعمال شرط انتقال امن در زمان‌بندی: به منظور آن که انتقال سامانه از حالت با بحرانیت پایین به حالت با بحرانیت بالا امن باشد باید در هنگام زمان‌بندی در حالت با بحرانیت بالای سامانه به ازای هر وظیفه τ و در هر زمان t شرط انتقال امن بررسی شود (VerifySafeTransition(τ, t)). به ازای هر وظیفه τ که اجرای آن تا زمان t در حالت با بحرانیت پایین سامانه تمام نشده است، اگر $et_\tau^{Low}(r_\tau, t+1) = et_\tau^{High}(r_\tau, t)$ باشد به این معنی است که اجرای وظیفه در بازه زمانی t تا $t+1$ در حالت با بحرانیت بالای سامانه، موجب پیشی گرفتن مقدار $et_\tau^{High}(r_\tau, t+1)$ از $et_\tau^{Low}(r_\tau, t+1)$ و نقض شرط انتقال امن در زمان $t+1$ خواهد شد. پس باید از اجرای وظیفه در بازه زمانی t تا $t+1$ ممانعت به عمل آید تا انتقال امن سامانه تضمین شود. با توجه به این که زمان‌بندی به صورت قبضه‌ای است شرط انتقال امن سامانه باید

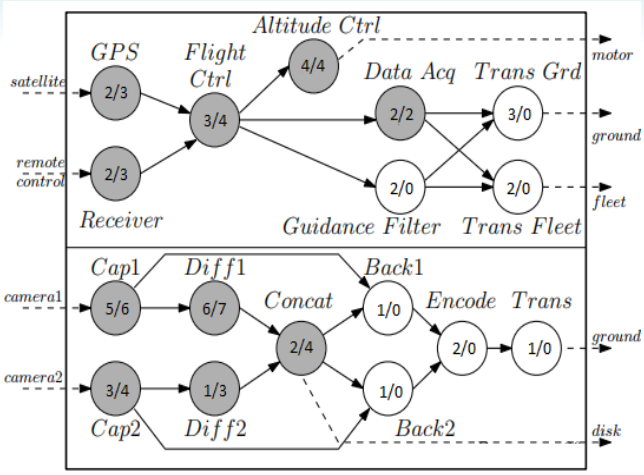
هم برای وظایف آماده به اجرا و هم برای وظایف در حال اجرا بررسی شود.

به منظور بررسی شرط انتقال امن برای وظایف آماده به اجرا، هنگام مرتب سازی وظایف موجود در $Ready_{list}$، ابتدا VerifySafeTransition(τ, t) برای هر وظیفه بررسی می‌شود و وظایف نقض کننده‌ی شرط، در لحظه t اجازه اجرا ندارند و به انتهای لیست می‌روند. باقی وظایف نیز بر اساس $laxity$ خود از ابتدای لیست مرتب می‌شوند. اگر بعد از مرتب سازی $Ready_{list}$ با الگوی گفته شده، اولین وظیفه لیست یک وظیفه‌ی نقض کننده‌ی شرط انتقال امن باشد به این معنیست که تمام وظایف لیست نقض کننده‌ی شرط انتقال امن هستند بنابراین هیچ‌کدام از این وظایف در لحظه t اجازه تخصیص ندارند و مرحله تخصیص در لحظه t به کل انجام نخواهد شد.

به منظور بررسی شرط انتقال امن برای وظایف در حال اجرا، باید در مرحله به روز رسانی لیست وظایف در حال اجرا به ازای هر وظیفه VerifySafeTransition(τ, t) بررسی شود و اگر وظیفه، نقض‌کننده شرط بود آن وظیفه باید متوقف شود و ادامه اجرای آن به زمانی دیرتر موکول شود. در این صورت وظیفه از حال اجرا خارج شده و به وظایف آماده به اجرا می‌پیوندد. هم‌چنین هسته میزبان آن وظیفه نیز به حالت بی‌کار در می‌آید و جدول زمان‌بندی نیز با توجه به متوقف شدن این وظیفه به روز رسانی می‌شود.

۴- نمونه انگیزشی

در این بخش، یک نمونه زمان‌بندی حل شده به وسیله الگوریتم شرح داده شده در این مقاله نمایش داده می‌شود. یک پهپاد یا هواپیمای بدون سرنشین (UAV[1]) به عنوان یک سامانه نمونه در نظر گرفته می‌شود. نمایش گرافیکی نرم‌افزار این سامانه در شکل (۲) ارائه شده است. [۱۴]



شکل (۲): سامانه UAV ([۱۴])

سامانه UAV مورد نظر از دو بخش تشکیل شده است. بخش اول (بالای شکل) مرتبط با سیستم کنترل پرواز (FCS[2]) است [۱۹] که وظایف مرتبط با این بخش در یک DAG نمایش داده شده است و دروه تناوب آن برابر با $T_{FCS} = 12$ است. بخش دوم (پایین شکل) سیستم پردازش تصویر با نام Montage است [۲۰] که وظایف مرتبط با این بخش در یک DAG نمایش داده شده است و دروه تناوب آن برابر با $T_{Montage} = 24$ است.

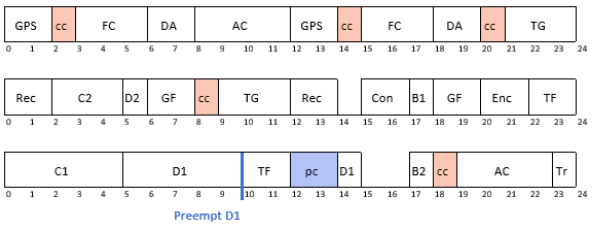
هر کدام از این DAG ها از تعدادی گره تشکیل شده‌اند که نشان‌دهنده وظایف هستند. گره‌های با رنگ سفید نشان‌دهنده وظایف با سطح بحرانیت پایین و گره‌های با رنگ خاکستری نشان‌دهنده وظایف با سطح بحرانیت بالا هستند. در داخل هر گره سهمیه زمانی آن وظیفه در حالت با بحرانیت پایین و حالت با بحرانیت بالای سامانه نمایش داده شده است. (عدد اول از سمت چپ برابر با $C(Low)$ و عدد دوم برابر با $C(High)$ آن وظیفه است) وظایف با سطح بحرانیت پایین در حالت با بحرانیت بالای سامانه شرکت ندارند بنابراین سهمیه زمانی آن‌ها در حالت با بحرانیت بالای سامانه برابر با صفر است. هر DAG علاوه بر گره‌ها، شامل تعدادی یال است که نشان‌دهنده وابستگی داده‌ای بین وظایف است . پیکان‌های نقطه‌چین نیز نشان‌دهنده آن است که داده‌ها از کجا آمده‌اند یا به کجا فرستاده می‌شوند.

زمان‌بندی این سامانه می‌بایست در یک Hyper-Period بررسی شود. Hyper-Period این سامانه برابر با ۲۴ خواهد بود. بنابراین دو فعالسازی از وظایف موجود در FCS و یک فعالسازی از وظایف موجود در Montage خواهیم داشت.

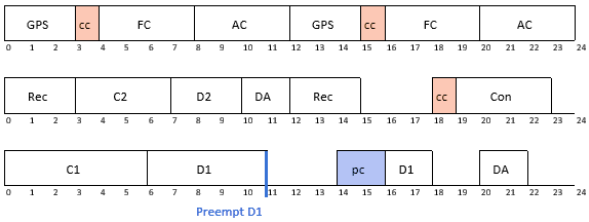
[1] Unmanned Aerial Vehicle

[2] Flight Control System

زمان‌بندی وظایف این سامانه با استفاده از الگوریتم این مقاله و با فرض داشتن یک معماری ۳ هسته‌ای و $PF = CF = 0.4$ انجام می‌شود . نتیجه زمان‌بندی این سامانه در حالت با بحرانیت پایین در شکل (۳) و نتیجه زمان‌بندی در حالت با بحرانیت بالا در شکل (۴) آمده است. دراین زمان‌بندی‌ها، قسمت‌های با عنوان cc که قبل از اجرای برخی وظایف وجود دارد نشان‌دهنده هزینه ارتباط تحمیل‌شده به آن وظیفه است و قسمت‌های با عنوان pc که قبل از اجرای برخی وظایف وجود دارد نشان‌دهنده هزینه قبضه تحمیل‌شده به آن وظیفه است. همان‌طور که مشخص است تعداد قبضه‌ها در حالت با بحرانیت پایین سامانه برابر با ۱ و در حالت با بحرانیت بالای سامانه هم برابر با ۱ است.



شکل (۳): زمان‌بندی UAV در حالت با بحرانیت پایین سامانه



شکل (۴): زمان‌بندی UAV در حالت با بحرانیت بالای سامانه

۵- نتایج آزمایشات

در این بخش نتایج و آزمایش‌های انجام شده به منظور ارزیابی زمان‌بندی ارائه شده، شرح داده می‌شوند.

۵-۱- محیط آزمایش شبیه‌سازی

به منظور انجام آزمایشات، نیاز به یک ابزار برای تولید تصادفی MCS و متشکل از MC-DAG هایی با تنظیمات مختلف است. این ابزار قبلا توسط [۱۴] پیاده‌سازی شده است که در این پژوهش هم از آن استفاده خواهد شد. این ابزار تضمین می‌کند که MCS تولید شده بی‌طرفانه باشد و به طور یکنواخت تنظیمات احتمالی زمان بندی را پوشش دهد. در این ابزار، به منظور تولید بی‌طرفانه از روش‌های موجود در [۲۱] برای تولید DAG با توپولوژی‌های بی‌طرفانه استفاده شده است و به منظور توزیع یکنواخت بهره‌برداری وظایف از ادغام روش‌های موجود در [۲۲] و [۲۳] استفاده شده است. مهم‌ترین پارامترهای این ابزار برای تولید یک MCS تصادفی به شرح زیر است:

- U_{MAX} که بیان‌گر بیشینه بهره‌برداری سامانه MCS در دو حالت با بحرانیت بالا و بحرانیت پایین است.
- ND که بیان‌گر تعداد MC-DAG های موجود در MCS است.
- NT که بیان‌گر تعداد MC-Task ها در هر MC-DAG است.
- e که احتمال وجود یال بین دو گره در یک MC-DAG را نشان می‌دهد. (احتمال وجود وابستگی بین دو وظیفه)

۵-۲- شرح آزمایش‌های انجام گرفته

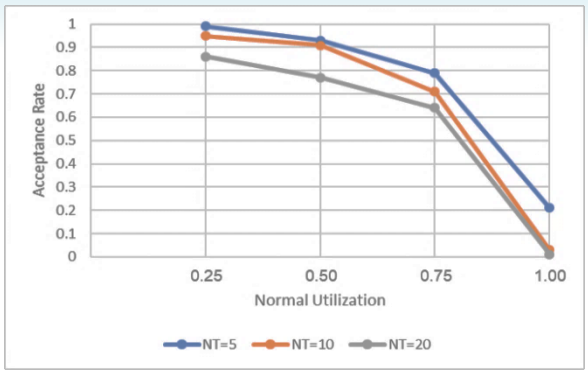
در این بخش با انجام آزمایشات مختلف، ابتدا تاثیر پارامترهای مختلف در زمان‌بندی ارائه شده توسط این مقاله تحلیل می‌گردد و پس از آن با الگوریتم [۱۴] (قبضه‌ای) مقایسه می‌گردد.

۵-۲-۱- تحلیل پارامترهای مختلف در زمان‌بندی ارائه شده

هدف از انجام آزمایش‌ها، ارزیابی نرخ زمان‌بندپذیری یا نرخ پذیرش[1] زمان‌بندی ارائه شده در این مقاله است. نرخ پذیرش یک زمان‌بندی برابر با نسبت آزمایش‌های زمان‌بندپذیر توسط آن زمان‌بندی به کل آزمایش‌های انجام شده است.

در شکل (۵) یک نمونه آزمایش انجام شده به منظور ارزیابی نرخ پذیرش نمایش داده شده است. محور افقی نشان‌دهنده بهره‌برداری نرمال سامانه و برابر با $U_{Norm} = U_{MAX}/m$ است. U_{MAX} همان بیشینه بهره‌برداری سامانه و m تعدادهسته‌های سامانه به هنگام زمابنندی است. محور عمودی نیز نشان‌دهنده نرخ پذیرش است. هم‌چنین در هر نقطه از نمودار، ۱۰۰ نمونه تصادفی MCS تولید و به وسیله الگوریتم آزمایش شده است. در این آزمایش به منظور تولید MCS های تصادفی، از پارامترهای $ND = 2$ ، $e = 20\%$ و $NT = 5, 10, 20$ استفاده شده است. هم‌چنین در زمان‌بندی $m = 4$ و $CF = 0.2$ و $PF = 0.2$ در نظر گرفته شده است.

[1] Acceptance Rate

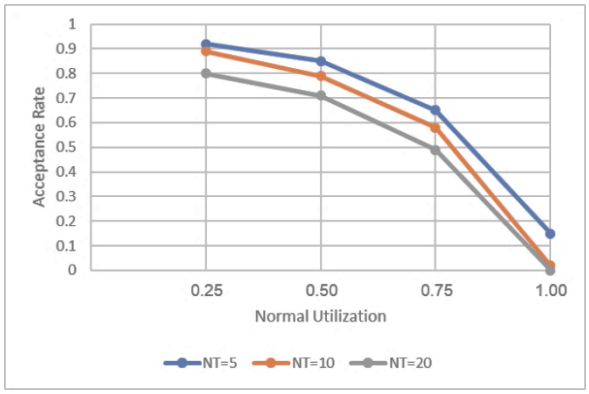


شکل (۵): نتایج زمان‌بندپذیری با تنظیمات ND=2, e=20%, m=4, CF=0.2, PF=0.2

تاثیر U_{MAX}: همان‌طور که در شکل (۵) مشخص است با حرکت در راستای محور افقی و افزایش U_{MAX} (افزایش U_{MAX} متناظر با افزایش بهره‌برداری نرمال است) نرخ زمان‌بندپذیری کاهش می‌یابد. در واقع افزایش U_{MAX} باعث می‌شود تا سامانه متراکم‌تر و محدودیت‌های زمانی وظایف افزایش یابد و بنابراین زمان‌بندی در مهلت مقرر هر وظیفه سخت‌تر شود.

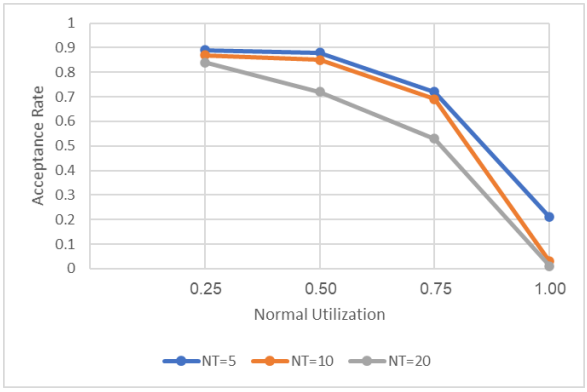
تاثیر NT: همان‌طور که در شکل (۵) مشخص است افزایش تعداد وظایف موجب کاهش در نرخ زمان‌بندپذیری شده است. با افزایش تعداد وظایف، در هر لحظه از زمان‌بندی سامانه، تعداد وظایف آماده برای زمان‌بندی در آن لحظه افزایش می‌یابد و از آن‌جایی که تعداد منابع پردازشی ثابت مانده است بنابراین احتمال از دست رفتن مهلت وظایف بالا می‌رود.

تاثیر ND: به منظور بررسی تاثیر تعداد DAG ها در زمان‌بندپذیری، در شکل (۶) تعداد DAG ها را نسبت به شکل (۵) افزایش داده و $ND = 4$ قرار گرفته است. افزایش تعداد DAG ها، موجب افزایش تعداد وظایف می‌گردد و بنابراین به همان دلیل بیان‌شده در مورد قبل، موجب کاهش نرخ زمان‌بندپذیری نسبت به شکل (۵) شده است.



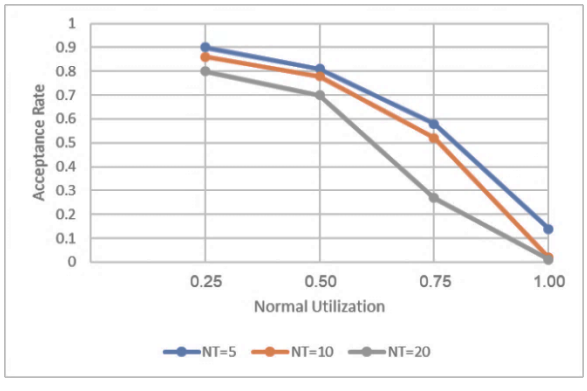
شکل (۶): نتایج زمان‌بندپذیری با تنظیمات ND=4, e=20%, m=4, CF=0.2, PF=0.2

تاثیر e: به منظور بررسی تاثیر افزایش وابستگی‌های بین وظایف در زمان‌بندپذیری، در شکل (۷)، e را نسبت به شکل (۵) افزایش داده و $e = 40\%$ قرار گرفته است. افزایش e ، موجب افزایش وابستگی بین وظایف و در نتیجه اعمال محدودیت‌های بیشتر در زمان‌بندی وظایف می‌گردد و بنابراین موجب کاهش نرخ زمان‌بندپذیری نسبت به شکل (۵) شده است.



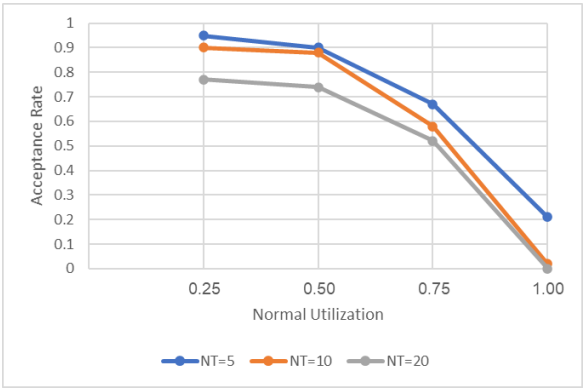
شکل (۷): نتایج زمان‌بندپذیری با تنظیمات ND=2, e=40%, m=4, CF=0.2, PF=0.2

تاثیر m: به منظور بررسی تاثیر تعداد هسته‌ها در زمان‌بندپذیری، در شکل (۸) تعداد هسته‌ها را نسبت به شکل (۵) افزایش داده و $m = 8$ قرار گرفته است. همان‌طور که مشخص است افزایش تعداد هسته‌ها موجب کاهش نرخ زمان‌بندپذیری نسبت به شکل (۵) شده است (بهره‌برداری نرمال در نقاط متناظر یکسان است و m دو برابر شده است بنابراین باتوجه به رابطه $U_{Norm} = U_{MAX}/m$ ، U_{MAX} نیز دو برابر شده است و از آن‌جایی که افزایش U_{MAX} موجب کاهش نرخ زمانبندپذیری می‌شود در این‌جا هم با کاهش نرخ زمانبندپذیری روبه‌رو هستیم).



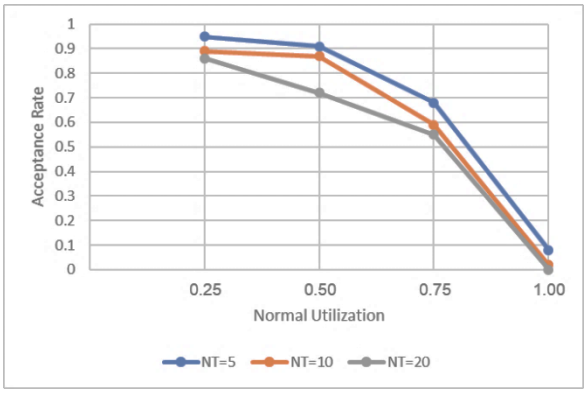
شکل (۸): نتایج زمان‌بندپذیری با تنظیمات ND=2, e=20%, m=8, CF=0.2, PF=0.2

تاثیر CF : برای بررسی تاثیر اندازه CF در زمان‌بندپذیری، در شکل (۹) مقدار CF را نسبت به شکل (۵) افزایش داده و $CF = 0.4$ قرار گرفته است. افزایش CF باعث می‌شود که هزینه ارتباط بیش‌تری به عنوان سربار به وظایف وابسته اضافه شود و زمان‌بندی وظایف در مهلت مقرر سخت‌تر شود. بنابراین نرخ زمان‌بندپذیری نسبت به شکل (۵) کاهش یافته است.



شکل (۹): نتایج زمان‌بندپذیری با تنظیمات ND=2, e=20%, m=4, CF=0.4, PF=0.2

تاثیر PF : برای بررسی تاثیر PF در زمان‌بندپذیری، در شکل (۱۰) مقدار PF را نسبت به شکل (۵) افزایش داده و $PF = 0.4$ قرار گرفته است. افزایش PF باعث می‌شود که هزینه قبضه بیش‌تری به عنوان سربار به وظایف در هنگام شروع مجدد بعد از قبضه شدن اضافه شود و زمان‌بندی وظایف در مهلت مقرر سخت‌تر شود. بنابراین نرخ زمان‌بندپذیری نسبت به شکل (۵) کاهش یافته است.

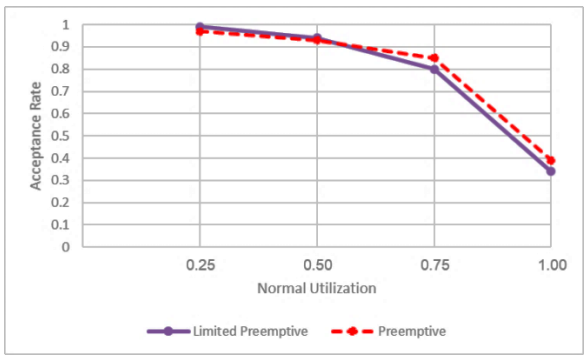


شکل (۱۰): نتایج زمان‌بندپذیری با تنظیمات ND=2, e=20%, m=4, CF=0.2, PF=0.4

۵-۲-۲- مقایسه با روش قبضه‌ای

زمان‌بندی در این پژوهش به صورت قبضه‌ای محدود ارائه شد و در این بخش با الگوریتم [۱۴] (قبضه‌ای) که بیشترین تشابه را از نظر تعریف مساله با این پژوهش دارد مقایسه می‌گردد. در الگوریتم قبضه‌ای، هزینه قبضه و ارتباط لحاظ نشده است. بنابراین $PF = CF = 0$ در نظر گرفته می‌شود. در این بخش، زمان‌بندپذیری و فراوانی قبضه ($Preemption\ Frequency = \frac{total\ preemptions}{hyper-period}$) دو الگوریتم مقایسه می‌گردند.

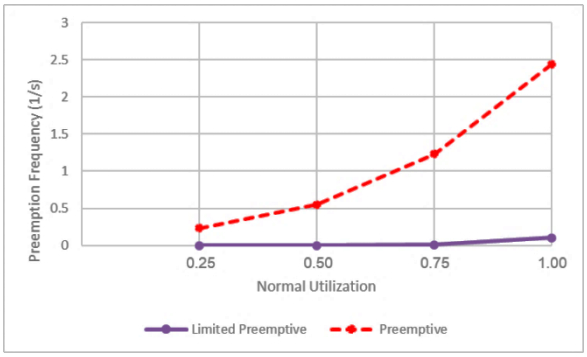
در شکل (۱۱) یک نمونه آزمایش انجام شده به منظور مقایسه نرخ زمان‌بندپذیری نمایش داده شده است. محور افقی نشان‌دهنده بهره‌برداری نرمال سامانه و محور عمودی نشان‌دهنده نرخ پذیرش الگوریتم است. در هر نقطه از نمودار، ۱۰۰ نمونه تصادفی MCS تولید و به وسیله الگوریتم این پژوهش (قبضه‌ای محدود) و الگوریتم قبضه‌ای آزمایش شده است. در این آزمایش به منظور تولید MCS های تصادفی، از پارامترهای $ND = 2$ ، $e = 20\%$ و $NT = 10$ استفاده شده است. هم‌چنین حین زمان‌بندی $m = 4$ و $PF = CF = 0$ در نظر گرفته شده است. همان‌طور که در شکل (۱۱) مشخص است با وجود این‌که در الگوریتم ارائه شده در این پژوهش قبضه به صورت محدود و کنترل شده است ولی با این حال نرخ زمان‌بندپذیری کاهش محسوسی نسبت به الگوریتم قبضه‌ای نداشته است.



شکل (۱۱): مقایسه زمان‌بندپذیری دو الگوریتم با تنظیمات ND=2, NT=10, e=20%, m=4, CF=PF=0

در شکل (۱۲) یک نمونه آزمایش انجام شده به منظور مقایسه فراوانی قبضه نمایش داده شده است. محور افقی نشان‌دهنده بهره‌برداری نرمال سامانه و محور عمودی نشان‌دهنده فراوانی قبضه الگوریتم است. در هر نقطه از نمودار، ۱۰۰ نمونه تصادفی MCS که زمان‌بندپذیر هستند توسط الگوریتم این پژوهش (قبضه‌ای محدود) و الگوریتم قبضه‌ای آزمایش شده‌اند و متوسط فراوانی قبضه این نمونه‌ها در نمودار نمایش داده شده است. به منظور تولید MCS های تصادفی، از پارامترهای $ND = 2$ ، $e = 20\%$ و $NT = 10$ استفاده شده است. هم‌چنین حین زمان‌بندی $m = 4$ و $PF = CF = 0$ در نظر گرفته

شده است. همان‌طور که در شکل (۱۲) مشخص است فراوانی قبضه در الگوریتم ارائه شده در این پژوهش در برابر فراوانی قبضه در الگوریتم قبضه‌ای بسیار ناچیز است.



شکل (۱۲): مقایسه فراوانی قبضه دو الگوریتم با تنظیمات ND=2, NT=10, e=20%, m=4, CF=PF=0

۶- نتیجه

در این پژوهش، زمان‌بندی وظایف وابسته و متناوب در سامانه‌های بحرانی-مختلط با معماری چندهسته‌ای مورد بررسی قرار گرفت. در سایر پژوهش‌های انجام شده در این حوزه، به نکاتی چون کاهش قبضه‌ها و تاثیر هزینه قبضه و ارتباط در زمان‌بندی توجه نشده بود و در این پژوهش درکنار حل مساله زمان‌بندی، این نکات نیز در نظر گرفته شدند. الگوریتم زمان‌بندی با پذیرش انجام قبضه در مواقع ضروری، به صورت قبضه‌ای محدود ارائه گردید. زمان‌بندی به روش استاتیک و دارای خروجی جداول زمان‌بندی در حالت با بحرانیت پایین و بالا است. الگوریتم ارائه شده از لحاظ تاثیر تنظیمات مختلف در نرخ زمان‌بندپذیری مورد آزمایش قرار گرفت. افزایش تعداد وظایف، تعداد DAGها، تعداد هسته‌ها، CF، PF ، تعداد یال‌ها و بهره‌برداری سامانه موجب کاهش نرخ زمان‌بندپذیری می‌شود. هم‌چنین این الگوریتم در مقایسه با الگوریتم قبضه‌ای بدون آن‌که تغییر محسوسی در نرخ زمان‌بندپذیری نشان دهد تعداد قبضه‌ها را به صورت چشمگیری کاهش داد.

در راستای تکمیل مطالب این پژوهش می‌توان در پژوهش‌های آتی به مسائلی چون افزایش QoS[1] وظایف با سطح بحرانیت پایین در حالت با بحرانیت بالای سامانه یا اعمال هزینه انتقال از حالت با بحرانیت پایین به حالت با بحرانیت بالای سامانه پرداخت.

[1] Quality of Service

ضمایم

- شبه کد الگوریتم محاسبه جدول زمان‌بندی در حالت با بحرانیت پایین سامانه

```
1.  Input S : MCS to schedule limited preemptive in LOW Mode
2.  Ready_list ← ∅
3.  Running_list ← ∅
4.  Idle_list ← {c ∈ Π}
5.  for all graph G_i ∈ Γ do
6.        ∀τ_j ∈ G_i : remaining(τ_j) ← C_j(Low)
7.        Ready_list = Ready_list ∪ {τ_j ∈ V_i | pred(τ_j) = ∅}
8.  end for
9.  for all timeslots 0 ≤ t < HP(S) do
10.       Sort τ ∈ Ready_list by laxity(τ,t)
11.       while Ready_list ≠ ∅ do
12.             τ_x ← Ready_list[0]
13.             if Idle_list ≠ ∅ then
14.                   c_x ← (c ∈ Idle_list | c minimize communications)
15.             else if laxity(τ_x, t) = 0 and max_{τ∈Running_list} laxity(τ,t) > 0 then
16.                   τ_y ← (τ ∈ Running_list | τ maximize laxity(τ,t))
17.                   c_x ← (c | table^Low[t,c] = τ_y or table^Low[t,c] = τ_y_load)
18.                   t' ≥ t : table^Low[t', c_x] ← ∅
19.                   Running_list ← Running_list − {τ_y}
20.                   Ready_list ← Ready_list ∪ {τ_y}
21.             else
22.                   break while
23.             end if
24.             if remaining(τ_x) < C_x(Low) then
25.                   load(τ_x) ← preemption_cost(τ_x)
26.             else
27.                   load(τ_x) ← max_{τ_p ∈ pred(τ_x)} communication_cost(τ_p, τ_x)
28.             end if
29.             if not VerifyDeadline(τ_x, t) then
30.                   return Not Schedulable
31.             end if
32.             t_x ∈ [t, t + load(τ_x)[ : table^Low[t_x, c_x] ← τ_x_load
33.             t_x ∈ [t + load(τ_x), t + load(τ_x) + remaining(τ_x)[: table^Low[t_x, c_x] ← τ_x
34.             Idle_list ← Idle_list − {c_x}
35.             Ready_list ← Ready_list − {τ_x}
36.             Running_list ← Running_list ∪ {τ_x}
37.       end while
38.       for all task τ ∈ Running_list do
39.             if load(τ) ≠ 0 then
40.                   load(τ) ← load(τ) − 1
41.             else
42.                   remaining(τ) ← remaining(τ) − 1
43.             end if
44.             if remaining(τ) = 0 then
45.                   Ready_list = Ready_list ∪
                        {succ(τ) | ∀τ_p ∈ pred(succ(τ)), remaining(τ_p) = 0}
46.                   Running_list ← Running_list − {τ}
47.                   Idle_list ← Idle_list ∪ {c | table^Low[t,c] = τ}
48.             end if
49.       end for
50.       for all G_i ∈ Γ do
51.       if (t + 1) mode T_i = 0 then
52.             ∀τ_j ∈ V_i : remaining(τ_j) ← C_j(Low)
53.             Ready_list = Ready_list ∪ {τ_j ∈ V_i | pred(τ_j) = ∅}
54.       end if
55.       end for
56. end for
57. return table^Low
```

- شبه کد الگوریتم محاسبه جدول زمان‌بندی در حالت با بحرانیت بالای سامانه

```
1.  Input S : MCS to schedule limited preemptive in HIGH Mode
2.  Remove LOW tasks and their edges
3.  Ready_list ← ∅
4.  Running_list ← ∅
5.  Idle_list ← {c ∈ Π}
6.  for all graph G_i ∈ Γ do
7.        ∀τ_j ∈ G_i : remaining(τ_j) ← C_j(High)
8.        Ready_list = Ready_list ∪ {τ_j ∈ V_i | pred(τ_j) = ∅}
9.  end for
10. for all timeslots 0 ≤ t < HP(S) do
11. Sort τ ∈ Ready_list by laxity(τ,t) and VerifySafeTransition(τ,t)
12.       while Ready_list ≠ ∅ do
13.             if not VerifySafeTransition(Ready_list[0], t) then
14.                   break while
15.             end if
16.             τ_x ← Ready_list[0]
17.             if Idle_list ≠ ∅ then
18.                   c_x ← (c ∈ Idle_list | c minimize communications)
19.             else if laxity(τ_x, t ) = 0 and max_{τ∈Running_list} laxity(τ,t) > 0 then
20.                   τ_y ← (τ ∈ Running_list | τ maximize laxity(τ,t))
21.                   c_x ← (c | table^High[t,c] = τ_y or table^High[t,c] = τ_y_load)
22.                   t' ≥ t : table^High[t', c_x] ← ∅
23.                   Running_list ← Running_list − {τ_y}
24.                   Ready_list ← Ready_list ∪ {τ_y}
25.             else
26.                   break while
27.             end if
28.             if remaining(τ_x) < C_x(High) then
29.                   load(τ_x) ← preemption_cost(τ_x)
30.             else
31.                   load(τ_x) ← max_{τ_p ∈ pred(τ_x)} communication_cost(τ_p, τ_x)
32.             end if
33.             if not VerifyDeadline(τ_x, t) then
34.                   return Not Schedulable
35.             end if
36.             t_x ∈ [t, t + load(τ_x)[ : table^High[t_x, c_x] ← τ_x_load
37.             t_x ∈ [t + load(τ_x), t + load(τ_x) + remaining(τ_x)[ :table^High[t_x, c_x] ← τ_x
38.             Idle_list ← Idle_list − {c_x}
39.             Ready_list ← Ready_list − {τ_x}
40.             Running_list ← Running_list ∪ {τ_x}
41.       end while
42.       for all task τ ∈ Running_list do
43.             if load(τ) ≠ 0 then
44.                   load(τ) ← load(τ) − 1
45.             else
46.                   remaining(τ) ← remaining(τ) − 1
47.             end if
48.             if remaining(τ) = 0 then
49.                   Ready_list = Ready_list ∪
                        {succ(τ) | ∀τ_p ∈ pred(succ(τ)), remaining(τ_p) = 0}
50.                   Running_list ← Running_list − {τ}
51.                   Idle_list ← Idle_list ∪ {c | table^High[t,c] = τ}
52.             end if
53.             if not VerifySafeTransition(τ,t) then
54.                   c' ← (c | table^High[t,c] = τ )
55.                   t' ≥ t : table^High[t', c'] ← ∅
56.                   Running_list ← Running_list − {τ}
57.                   Ready_list ← Ready_list ∪ {τ}
58.                   Idle_list ← Idle_list ∪ {c'}
59.             end if
60.       end for
61.       for all G_i ∈ Γ do
62.             if (t + 1) mode T_i = 0 then
63.                   ∀τ_j ∈ V_i : remaining(τ_j) ← C_j(High)
64.                   Ready_list = Ready_list ∪ {τ_j ∈ V_i | pred(τ_j) = ∅}
65.             end if
66.       end for
67. end for
68. return table^High
```

مراجع

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE international real-time systems symposium (RTSS 2007)*, 2007: IEEE, pp. 239-243.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1-69, 2013.

[3] S. Baruah, "Mixed-Criticality Scheduling Theory: Scope, Promise, and Limitations," *IEEE Des. Test*, vol. 35, no. 2, pp. 31-37, 2018.

[4] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011.

[5] S. Baruah, "Implementing mixed-criticality synchronous reactive programs upon uniprocessor platforms," *Real-Time Systems*, vol. 50, no. 3, pp. 317-341, 2014.

[6] S. Baruah, "Semantics-preserving implementation of multirate mixed-criticality synchronous programs," in *Proceedings of the 20th International Conference on Real-Time and Network Systems*, 2012, pp. 11-19.

[7] S. Baruah, "Implementing mixed criticality synchronous reactive systems upon multiprocessor platforms," *The University of North Carolina at Chapel Hill, Tech. Rep*, 2013.

[8] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Springer, 2017.

[9] R. Medina, E. Borde, and L. Pautet, "Directed acyclic graph scheduling for mixed-criticality systems," in *Ada-Europe International Conference on Reliable Software Technologies*, 2017: Springer, pp. 217-232.

[10] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic DAG tasks," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016: IEEE, pp. 227-236.

[11] T. Yang, Y. Tang, X. Jiang, Q. Deng, and N. Guan, "Semi-Federated Scheduling of Mixed-Criticality System for Sporadic DAG Tasks," in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, 2019: IEEE, pp. 163-170.

[12] R. M. Pathan, "Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, 2018: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[13] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-time systems*, vol. 53, no. 5, pp. 760-811, 2017.

[14] R. Medina, E. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality dags on multi-core architectures," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018: IEEE, pp. 254-264.

[15] R. Medina, E. Borde, and L. Pautet, "Generalized Mixed-Criticality Static Scheduling for Periodic Directed Acyclic Graphs on Multi-Core Processors," *IEEE Transactions on Computers*, 2020.

[16] S. Baruah *et al.*, "Scheduling real-time mixed-criticality jobs," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140-1152, 2011.

[17] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012: IEEE, pp. 166-175.

[18] H.-E. Zahaf, G. Lipari, and S. Niar, "Preemption-Aware Allocation, Deadline Assignment for Conditional DAGs on Partitioned EDF," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2020: IEEE, pp. 1-10.

[19] S. Siebert and J. Teizer, "Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system," *Automation in construction*, vol. 41, pp. 1-14, 2014.

[20] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 third workshop on workflows in support of large-scale science*, 2008: IEEE, pp. 1-10.

[21] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*, 2010: ICST, p. 10.

[22] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129-154, 2005.

[23] R. I. Davis and A. Burns, "Priority assignment for