



دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

پایان نامه کارشناسی

پیاده‌سازی چارچوبی برای زمان‌بندی سیستم‌های بحرانی مختلط

نگارنده

مریم علی‌کرمی

استاد راهنما

دکتر حمیدرضا زرندي

۱۴۰۳ فروردین



اینجانب مریم علی کرمی متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی استادی 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 مأخذ 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 مأخذ بلامنع است.

مریم علی کرمی

امضا

## سپاس‌گزاری

در اینجا لازم می‌دانم که از استاد گرانقدر، جناب آقای دکتر حمیدرضا زرندی، که بدون راهنمایی‌های ارزشمند ایشان انجام این پروژه میسر نمی‌شد، نهایت تشکر را داشته باشم. همچنین از استاد گران‌مایه، جناب آقای دکتر احمد جوادی، که زحمت داوری این پایان‌نامه را برعهده داشتند، نهایت تشکر را دارم.

مریم علی‌کرمی

۱۴۰۳ فروردین

## چکیده

با تمايل روزافرون صنعت به مجتمع‌سازی<sup>۱</sup> بستر اجرای وظایف با درجه‌ی بحرانی متفاوت، سامانه‌های بحرانی مختلط<sup>۲</sup> به عنوان نسل جدیدی از سامانه‌های بی‌درنگ<sup>۳</sup> بحرانی، معرفی شدند. اين مطالعه ضمن تعريف و بررسی سير تکاملی اين سامانه‌ها، با تمرکز بر رویکرد بهبود یافته زمان‌بندی<sup>۴</sup> بر اساس زودترین ضرب‌الاجل اول با مهلت مجازی<sup>۵</sup> (EDF-VD)، بررسی دقیقی از الگوریتم‌های زمان‌بندی در سامانه‌های بحرانی مختلط ارائه می‌دهد. هدف اصلی اين کار، ارزیابی عملکرد و قابلیت اطمینان EDF-VD در مدیریت وظایف با سطوح بحرانیت متفاوت است. مطالعه از طریق شبیه‌سازی زمان‌بندی سامانه‌های بحرانی مختلط بر روی پردازنده‌ای تک‌هسته‌ای با سرعت متغیر، نشان می‌دهد که EDF-VD در مقایسه با روش‌های سنتی، پیشرفت‌های قابل توجهی در مدیریت وظایف با بحرانیت مختلط ارائه می‌دهد. نتایج نشان می‌دهند که EDF-VD نه تنها اجرای به موقع وظایف با بحرانیت بالا را تضمین می‌کند، بلکه بهره‌وری کلی منابع را نیز بهبود می‌بخشد. این تحقیق با ارائه چارچوب زمان‌بندی مستحکمی که می‌تواند به طور بالقوه در سیستم‌های عامل واقعی برای زمان‌بندی برخط استفاده شود یا برای اجرای سامانه‌های بحرانی مختلط، یک زمان‌بندی ایستا تولید کند، به حوزه خود کمک می‌کند. همچنین، پیاده‌سازی منعطف، پیکربندی گستره و داده‌های آماری این چارچوب، امکان مقایسه‌ی تاثیر مولفه‌های گوناگون بر زمان‌بندی سامانه‌های بحرانی مختلط را فراهم می‌آورد. پیامدهای این یافته‌ها با پیشنهاداتی برای تحقیقات آینده در بهینه‌سازی زمان‌بندی با بحرانیت مختلط مورد بحث قرار می‌گیرد.

## واژه‌های کلیدی:

سامانه‌های بحرانی مختلط، سامانه‌های بی‌درنگ، زمان‌بندی سیستم‌های بحرانی مختلط، الگوریتم EDF-VD، تحلیل<sup>۶</sup> AMC، زمان‌بندی پردازنده تک‌هسته‌ای با سرعت متغیر، بررسی زمان‌بندی پذیری

<sup>۱</sup> Integration

<sup>۲</sup> Mixed Criticality Systems (MCS)

<sup>۳</sup> Realtime Systems

<sup>۴</sup> Scheduling

<sup>۵</sup> Earliest Deadline First – Virtual Deadline

<sup>۶</sup> Adaptive Mixed Criticality

## فهرست مطالب

صفحه	عنوان
۱	۱ - مقدمه
۶	۲ - معرفی مفاهیم پایه‌ی سامانه‌های بحرانی مختلط
۷	۲ - ۱ - سامانه‌های بی‌درنگ
۸	۲ - ۱ - ۱ - وظایف سامانه‌های بی‌درنگ
۹	۲ - ۱ - ۲ - زمان‌بندی سامانه‌های بی‌درنگ
۱۱	۲ - ۲ - سامانه‌های بحرانی مختلط
۱۴	۲ - ۲ - ۱ - مدل سامانه‌های بحرانی مختلط
۱۵	۲ - ۲ - ۲ - چالش‌های پیاده‌سازی
۱۶	۲ - ۲ - ۳ - چالش‌های صنعتی‌سازی
۱۸	۳ - زمان‌بندی سامانه‌های بحرانی مختلط
۱۹	۳ - ۱ - چالش‌های زمان‌بندی سامانه‌های بحرانی مختلط
۲۰	۳ - ۲ - زمان‌بندی اولویت ثابت با روش‌های مبتنی بر تحلیل زمان پاسخگویی
۲۰	۳ - ۲ - ۱ - الگوریتم CrMPO
۲۱	۳ - ۲ - ۲ - تحلیل چند-بحرانیتی وستال و ناکارآمدی الگوریتم DM
۲۳	۳ - ۲ - ۳ - روش‌های AMC و درجه‌ی بحرانیت یک رفتار
۲۹	۳ - ۳ - زمان‌بندی اولویت پویا با الگوریتم EDF
۲۹	۳ - ۳ - ۱ - اثبات ناکارآمدی EDF
۳۰	۳ - ۳ - ۲ - تحقیقات انجام‌شده برای گسترش EDF
۳۲	۳ - ۳ - ۳ - ۳ - تبدیل EDF به EDF-VD
۳۵	۳ - ۳ - ۴ - مقایسه‌ی الگوریتم EDF-VD با سایر الگوریتم‌ها
۳۷	۳ - ۳ - ۵ - محدودیت‌های EDF-VD
۳۹	۴ - پیاده‌سازی چارچوب زمان‌بندی
۴۰	۴ - ۱ - مدل سامانه بحرانی مختلط
۴۱	۴ - ۲ - جریان اجرای نرم‌افزارهای تولید وظایف و شبیه‌سازی زمان‌بندی

۴۴	۳ - ۴ - ابزارهای استفاده شده.....
۴۶	۴ - ۴ - پیاده‌سازی تولید مجموعه وظایف.....
۴۸	۴ - ۵ - پیاده‌سازی شبیه‌سازی زمان‌بندی.....
۴۹	۴ - ۵ - ۱ - متغیرهای سراسری.....
۵۱	۲ - ۴ - ۵ - کلاس شبیه‌ساز.....
۵۳	۴ - ۵ - ۳ - کلاس وظیفه.....
۵۳	۴ - ۵ - ۴ - کلاس کار.....
۵۶	۴ - ۵ - ۵ - کلاس پردازنده.....
۵۷	۴ - ۵ - ۶ - کلاس زمان‌بند.....
۵۷	۴ - ۵ - ۷ - کلاس صفات کارهای آماده.....
۵۸	۴ - ۵ - ۸ - کلاس تولید زمان اجرای حقيقی.....
۵۹	۴ - ۵ - ۹ - کلاس ذخیره‌کنندهی رخدادهای اجرا.....
۶۰	۴ - ۵ - ۱۰ - ارتباط بین کلاس‌ها.....
۶۱	۴ - ۶ - ساختار پوشش‌های نرم‌افزار.....
۶۲	۴ - ۷ - پیکربندی و دستورات اجرای نرم‌افزار.....
۶۴	۵ - ارزیابی پروژه.....
۶۵	۵ - ۱ - دریافت ورودی.....
۶۵	۵ - ۲ - تولید مجموعه وظایف.....
۶۷	۵ - ۳ - شبیه‌سازی زمان‌بندی.....
۶۷	۵ - ۳ - ۱ - زمان‌بندی سنتی تک‌حرانیتی روی پردازندهی تک‌هسته‌ای با سرعت واحد.....
۷۰	۵ - ۳ - ۲ - زمان‌بندی بحرانی مختلط با الگوریتم EDF-VD و مهلتهای واقعی.....
۷۴	۵ - ۳ - ۳ - زمان‌بندی بحرانی مختلط با الگوریتم EDF-VD و مهلتهای مجازی.....
۷۷	۴ - ۳ - ۵ - ورود اجباری سامانه به حالت عملیاتی بحرانی.....
۷۹	۵ - ۴ - محدودیت‌های نرم‌افزار.....
۸۱	۶ - جمع‌بندی، نتیجه‌گیری و پیشنهادات.....
۸۲	۶ - ۱ - جمع‌بندی و نتیجه‌گیری.....
۸۳	۶ - ۲ - پیشنهادها.....
۸۴	۷ - مراجع.....

فهرست شکل‌ها

عنوان	
صفحه	
۱ - انواع سیستم‌های بی‌درنگ	۷
۲ مقایسه‌ی انواع الگوریتم‌های زمان‌بندی اولویت ثابت در زمان‌بندی سامانه‌های بحرانی مختلط	۲۱
۳ - نمودار جریان اجرای شبیه‌ساز زمان‌بندی	۴۳
۴ - نماد ابزارهای استفاده شده برای پیاده‌سازی نرم‌افزار	۴۶
۵ - نمودار جریان اجرای نرم‌افزار تولید وظایف	۴۷
۶ - نمودار کلاس‌های نرم‌افزار شبیه‌سازی	۴۹
۷ - نمودار ارتباط بین کلاس‌های نرم‌افزار شبیه‌ساز زمان‌بندی	۶۱
۸ - نتایج کنترل ورودی‌های نرم‌افزار	۶۵
۹ - پیکربندی و اجرای نرم‌افزار تولید مجموعه وظایف	۶۶
۱۰ - مجموعه وظایف تولید شده توسط نرم‌افزار	۶۶
۱۱ - اجرای زمان‌بندی سنتی تک‌بحرانیتی در پردازنده با سرعت واحد	۶۷
۱۲ - زمان‌بندی تولید شده برای حالت سنتی بر روی پردازنده با سرعت واحد	۶۸
۱۳ - نتایج آماری اجرای زمان‌بندی سنتی بر پردازنده با سرعت واحد	۶۹
۱۴ - اجرای ناموفق زمان‌بندی سنتی بر مجموعه وظایف با بدترین زمان اجرای منتنسب به درجهی بحرانیت	۷۰
۱۵ - زمان‌بندی بحرانی مختلط با EDF-VD و مهلت‌های واقعی بر روی پردازنده با سرعت واحد	۷۱
۱۶ - زمان‌بندی بحرانی مختلط با EDF-VD و مهلت‌های واقعی بر روی پردازنده با سرعت ۱.۷	۷۲
۱۷ - اجرای زمان‌بندی EDF-VD با مهلت واقعی بر مجموعه وظایفی با احتمال تجاوز از زمان اجرای درصدی	۷۳
۱۸ - خروجی زمان‌بندی EDF-VD با مهلت واقعی بر مجموعه وظایفی با احتمال تجاوز از زمان اجرای درصدی	۷۳
۱۹ - مجموعه وظایف تولید شده برای زمان‌بندی با مهلت مجازی	۷۴
۲۰ - اجرای زمان‌بندی EDF-VD با مهلت مجازی بر روی [۲۰-۵] - ۰.۴۴ - ۰.۸ - ۴	۷۵

شکل ۲۱ - صفحه کارهای آماده در حین اجرای الگوریتم EDF-VD با مهلت مجازی بر روی [۵-۰.۴۴-۰.۸-۰.۴-۴] ..... ۷۶	20]
شکل ۲۲ - داده‌های آماری اجرای EDF-VD با مهلت مجازی بر روی [۵-۰.۴۴-۰.۸-۰.۴-۴] ..... ۷۶	
شکل ۲۳ - داده‌های آماری اجرای EDF-VD با مهلت مجازی بر روی [۵-۰.۴۴-۰.۸-۰.۴-۴] با احتمال صفر درصدی تجاوز از زمان اجرا ..... ۷۷	
شکل ۲۴ - پیکربندی سامانه برای ورود اجباری به حالت عملیاتی بحرانی در لحظه ۲ ..... ۷۸	
شکل ۲۵ - اجرای شبیه‌سازی برای ورود اجباری سامانه به حالت عملیاتی بحرانی در لحظه ۲ ..... ۷۸	
شکل ۲۶ - داده‌های آماری شبیه‌سازی ورود اجباری سامانه به حالت عملیاتی بحرانی در لحظه ۲ ..... ۷۹	

## فهرست جدول‌ها

عنوان		صفحه
جدول ۱ - سطوح بحرانی در استاندارد DO-178B	۲	.....
جدول ۲ - تعریف مولفه‌های وظیفه‌ی بحرانی مختلط	۱۴	.....
جدول ۴ - دسته‌بندی انواع سامانه‌های بحرانی مختلط اولویت ثابت	۲۳	.....
جدول ۵ - تحلیل زمان پاسخگویی SMC-NO و SMC	۲۵	.....
جدول ۶ - نمونه مجموعه وظایف بحرانی مختلط ارائه شده برای اثبات ناکارآمدی الگوریتم EDF	۳۰	.....
جدول ۷ - رخدادهای ذخیره شده در حین اجرای شبیه‌سازی	۵۹	.....
جدول ۸ - لیست فایل‌های پروژه	۶۱	.....

## ١ - مقدمة

با اقبال صنعت به سمت یکپارچه‌سازی بستر اجرای وظایف از درجه‌ی بحرانی متفاوت برای کاهش هزینه‌ها به جای استفاده از روش‌های بخش‌بندی منابع، سامانه‌های بحرانی مختلط به عنوان نسل جدیدی از سامانه‌های بی‌درنگ بحرانی معرفی شدند. این نوع از سامانه‌ها، وظایف با درجه‌ی بحرانی متفاوت را بر روی یک بستر سخت‌افزاری مشترک مجتمع‌سازی می‌کنند. سامانه‌های بحرانی مختلط در عین اطمینان از اجرای بدون مداخله و وظایف بحرانی‌ایمن<sup>۱</sup>، به کل مجموعه وظایف اجازه می‌دهند که از منابع سامانه به صورت اشتراکی استفاده و بر سر آن‌ها رقابت کنند.

درجه‌ی بحرانی هر وظیفه بر اساس میزان قابلیت اطمینان مورد نیاز برای آن وظیفه در استانداردهای صنعتی نظری DO-254، IEC 61508، ISO 26262 و DO-178B تعریف می‌شود. برای مثال، استاندارد DO-178B در صنعت الکترونیک هوایی<sup>۲</sup>، وظایف را مطابق جدول ۱ به ۵ سطح بحرانی تقسیم می‌کند [۱].

جدول ۱ - سطوح بحرانی در استاندارد DO-178B

سطح	وضعیت ناکارآمدی	تفسیر
A	فاجعه‌بار	ناکارآمدی ممکن است باعث خرابی شود.
B	خطرناک	ناکارآمدی تأثیر منفی زیادی بر بحران یا عملکرد دارد.
C	قابل توجه	ناکارآمدی قابل توجه است، اما تأثیر کمتری نسبت به شکست خطرناک دارد.
D	اندک	ناکارآمدی قابل توجه است، اما تأثیر کمتری نسبت به یک شکست بزرگ دارد.
E	بدون اثر	ناکارآمدی هیچ تأثیری بر ایمنی سامانه ندارد.

از آنجایی که ایمنی در این سامانه‌ها بسیار مهم است، سامانه‌های بی‌درنگ برای دریافت استانداردهای لازم باید به تایید مراجع صدور گواهینامه، به طور مخفف CA<sup>۳</sup>، برسند. برای اطمینان از اجرای صحیح و ایمن، CA برای تعیین بدترین زمان اجرای وظایف، به طور مخفف WCET<sup>۴</sup>، در سامانه‌های بحرانی، معیارهای سخت‌گیرانه‌ای دارد که عموماً منجر به تخمین مقادیر بسیار بزرگ‌تری از حد واقعی زمان اجرای وظایف می‌شود. در اینجا برای روشن‌سازی فضای مساله، مثالی را به صورت پرسش و پاسخ ارائه می‌دهیم.

<sup>۱</sup> Safety-Critical

<sup>۲</sup> Avionics

<sup>۳</sup> Certificate Authority

<sup>۴</sup> Worst Case Execution Time

**سوال ۱-۱:** سامانه‌ای را با دو وظیفه در نظر بگیرید:  $\tau_1$  که یک وظیفه بحرانی-ایمن و  $\tau_2$  که یک وظیفه بحرانی-غیر-ایمن است. هر دو وظیفه در زمان صفر منتشر می‌شوند و مهلتی برابر با ۱۰ دارند.  $WCET_i(o)$  بدترین زمان اجرای وظیفه‌ی  $\tau_i$  را مشخص می‌کند که مقدار آن توسط طراح سامانه تعریف شده است.  $WCET_i(p)$  بدترین زمان اجرای وظیفه‌ی  $\tau_i$  را مشخص می‌کند که مقدار آن توسط CA تعریف شده است. مقادیر  $WCET_2(p) = 4$ ,  $WCET_1(o) = 5$ ,  $WCET_1(p) = 5$ ,  $WCET_2(o) = 6$  را به سامانه اختصاص می‌دهیم. فرض می‌شود که  $\tau_1$  ابتدا اجرا شده و سپس  $\tau_2$  اجرا می‌شود. اگر سرعت پردازش این سامانه ۱ واحد در زمان باشد، آیا این سامانه، به گونه‌ای که بتواند نیازمندی‌های ایمنی صدور گواهی استاندارد و همینطور تضمین کیفیت را به طور همزمان در ۱۰ واحد زمانی پوشش دهد، قابل زمان‌بندی است؟

**پاسخ ۱-۱:** از دیدگاه CA، وظیفه  $\tau_1$  از لحظه‌ی ۰ تا لحظه ۵ اجرا و تکمیل می‌شود و از آنجایی که مهلت وظیفه ۱۰ است،  $\tau_1$  قبل از سرسید مهلت به پایان رسیده است. CA اهمیتی به انجام وظیفه  $\tau_2$  نمی‌دهد که آیا قبل از سرسید مهلت تکمیل شده است یا خیر، زیرا وظیفه  $\tau_2$  یک وظیفه بحرانی-غیر-ایمن است. بنابراین، مرجع صدور گواهینامه سامانه را ایمن تلقی می‌کند و به این عنوان برای سامانه گواهی صادر می‌کند؛ زیرا وظیفه بحرانی-ایمن می‌تواند قبل از مهلت تعیین شده تکمیل شود.

حال از دیدگاهی دیگر که طراح سامانه مشخص کرده و با مقادیر مدنظر او، سامانه را بررسی می‌کنیم. وظیفه  $\tau_1$  ابتدا شروع به کار می‌کند و اجرای آن را قبل از لحظه ۴ کامل می‌کند. پس از آن، وظیفه  $\tau_2$  شروع به کار کرده و تا لحظه ۹ کار خود را تکمیل می‌کند. از آنجا که مهلت هر دو وظیفه در لحظه‌ی ۱۰ فرا می‌رسد، هر دو با موفقیت و بدون از دست دادن هیچ مهلتی اجرا شده‌اند. بنابراین طراح سامانه نیز سامانه را صحیح اجرا کرده است. پس بله، این سامانه قابل زمان‌بندی است.

**سوال ۲-۱:** در ادامه‌ی سوال ۱-۱، فرض کنید که لازم باشد  $\tau_2$  قبل از  $\tau_1$  اجرا شود؛ برای مثال، مهلت آن در لحظه‌ی ۷ فرا می‌رسد. در چنین حالتی، اگر از نظر طراح سامانه به آن نگاه کنیم،  $\tau_2$  در لحظه‌ی ۵ و  $\tau_1$  در لحظه‌ی ۹ کامل می‌شوند و هر دو وظیفه به مهلت‌های خود می‌رسند. از نگاه CA اما، وظیفه‌ی ۲ در لحظه‌ی ۶ کامل می‌شود و فقط ۴ واحد زمانی برای  $\tau_1$  باقی می‌گذارد؛ پس چنین سامانه‌ای ایمن نیست و گواهی دریافت نمی‌کند چون  $\tau_1$  مهلت خود را از دست می‌دهد.

در چنین شرایطی برای اینکه بتوان سامانه را زمان‌بندی کرد و گواهی ایمنی آن را نیز دریافت کرد، باید ۱۱ واحد کار در ۱۰ واحد از زمان اجرا شود. یعنی، به توان پردازشی بیشتری نیاز داریم. آیا می‌شود بدون اختصاص این ظرفیت اضافه، زمان‌بندی سامانه را تضمین کرد؟

پاسخ ۱-۲: فرض کنید در سامانه سازوکاری تعبیه کنیم که در صورت اجرای  $\tau_2$  بیش از حد زمانی  $\tau_2$  را از سامانه حذف کند. در این صورت، اگر حق با طراح سامانه باشد،  $\tau_2$  در بازه‌ی زمانی ۰ تا ۵ اجرا خواهد شد و پس از آن ۵ واحد زمانی نیز برای اجرای  $\tau_1$  باقی می‌ماند که هم از نظر CA و هم از نظر طراح سامانه، برای اجرای وظیفه‌ی ۱ قبل از فرارسیدن مهلت آن، کافی است. در صورتی که حق با CA باشد و زمان اجرای  $\tau_2$ ، برابر با ۶ واحد باشد، سازوکار تعبیه‌شده در سامانه، در لحظه‌ی ۵،  $\tau_2$  را متوقف می‌کند و پس از آن نیز  $\tau_1$  زمان کافی برای اجرا خواهد داشت.

مدل سامانه‌های بحرانی مختلط معرفی شد تا به سوالاتی که در این مثال مطرح شد، پاسخ بدهد. وستال<sup>۱</sup> در سال ۲۰۰۷، این مدل را معرفی کرد و از آنجایی که  $WCET_1(p) + WCET_2(o) = 5 + 5 < 10$  نشان داد که این سامانه بدون اختصاص ظرفیت پردازشی اضافی نیز قابل زمان‌بندی می‌باشد و یک الگوریتم اختصاص اولویت و تحلیل زمان‌بندی برای این مدل سامانه‌ها ارائه کرد [۲]. پس از او افراد بسیار در زمینه‌ی سامانه‌های بحرانی مختلط به تحقیق و بررسی پرداختند و مدل معرفی شده توسط او را گسترش دادند که در این گزارش به معرفی آن‌ها و نتایج بررسی‌هایشان خواهیم پرداخت. چالش اصلی زمان‌بندی سامانه‌های بحرانی مختلط در ارائه‌ی الگوریتمی پوشاست که بتواند به سوالاتی از این قبیل، در ازای هر مجموعه وظایف دلخواه، پاسخ دهد.

هدف اصلی سامانه‌های بحرانی مختلط، حل مساله‌ای از جنس ایجاد تعادل میان دو جنبه‌ی متضاد کارایی و اطمینان است. این سامانه‌ها تلاش می‌کنند تا در عین حفظ اطمینان از اجرای صحیح و ایمن سامانه که در حد شدید با جداسازی وظایف از درجه‌ی بحرانی متفاوت محقق می‌شود، کارایی سامانه را نیز با مجتمع‌سازی وظایف و بهاشترانگذاری منابع، افزایش دهند [۳]. بحرانیت-مختلط یک مفهوم است که می‌تواند در سطح معماري، زمان‌بندی، تخصیص منابع و مدیریت خطای در سامانه اعمال شود. در این پروژه، تمرکز بر سطح زمان‌بندی و تخصیص پردازنده است.

هدف این پروژه، بعد از معرفی کامل سامانه‌های بحرانی مختلط و چالش‌های آن، پیاده‌سازی چارچوبی نرم‌افزاری برای تولید مجموعه وظایف بحرانی مختلط و زمان‌بندی آن با استفاده از الگوریتم نزدیک‌ترین ضرب‌الاجل با مهلت‌های مجازی، به طور مخفف EDF-VD، می‌باشد. EDF-VD و آزمون‌های زمان‌بندی آن توسط باروآح<sup>۲</sup>

<sup>۱</sup> Vestal

<sup>۲</sup> Baruah

همکارانش ابداع شدند. این الگوریتم یکی از برجسته‌ترین الگوریتم‌های زمان‌بندی سامانه‌های بحرانی مختلط با عملکردی بهتر از سایر الگوریتم‌های اولویت متغیر مطرح در این زمینه می‌باشد [۴].

به این منظور در فصل دوم، با تعریف سامانه‌های بی‌درنگ و الگوریتم‌های زمان‌بندی پایه، به معرفی پیش‌نیازهای شناخت سامانه‌های بحرانی مختلط می‌پردازیم. سپس نگاه عمیق‌تری به سامانه‌های بحرانی مختلط داریم و انواع این سامانه‌ها را به همراه مدل وظیفه و چالش‌های آن‌ها بر می‌شماریم. در فصل سوم، زمان‌بندی سامانه‌های بحرانی مختلط و روند پیشرفت الگوریتم‌های مطرح در این زمینه را با جزئیات کامل بررسی می‌کنیم؛ با الگوریتم‌های اولویت ثابت و آزمون‌های زمان‌بندی مبتنی بر تحلیل زمان پاسخگویی مانند<sup>۱</sup> AMC-rtb شروع می‌کنیم و در نهایت به EDF و گسترش آن به EDF-VD می‌رسیم و اثباتی را از عملکرد آن ارائه می‌دهیم. همچنین نشان می‌دهیم که الگوریتم‌های سنتی تک-بحرانیتی برای زمان‌بندی بحرانی مختلط مناسب نیستند.

در فصل چهارم، جزئیات پیاده‌سازی چارچوب نرم‌افزاری زمان‌بندی را با تعریف کلاس‌ها، روابط بین آن‌ها و توضیح جریان اجرا، شرح می‌دهیم. این نرم‌افزار از دو بخش تولید وظایف بحرانی مختلط و شبیه‌سازی زمان‌بندی آن‌ها تشکیل شده است و با استفاده از پیکربندی تنظیمات گسترده و پیاده‌سازی انعطاف‌پذیر، امکان مقایسه‌ی تاثیر مولفه‌های گوناگون بر زمان‌بندی سامانه را فراهم می‌کند. این شبیه‌سازی بر روی پردازنده‌ای تک‌هسته‌ای با سرعت متغیر اجرا می‌شود. در فصل پنجم، با ارائه‌ی اشکال و توضیحات، عملکرد این نرم‌افزار را ارزیابی می‌کنیم و محدودیت‌های آن را بر می‌شماریم. در فصل ششم نیز بعد از مرور بررسی‌های انجام شده، به جمع‌بندی و ارائه‌ی پیشنهاداتی برای آینده می‌پردازیم.

<sup>۱</sup> Adaptive Mixed Criticaliy – Response Time Bound

## ۲ - معرفی مفاهیم پایه‌ی سامانه‌های بحرانی مختلط

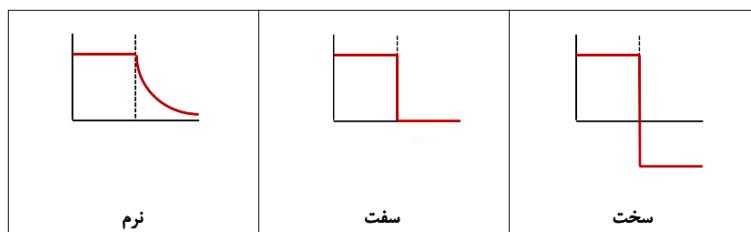
در این فصل به معرفی سامانه‌های بی‌درنگ و واژگان کاربردی آن‌ها می‌پردازیم و بعد از آن، سامانه‌های بحرانی مختلط را به عنوان نوعی از سامانه‌های بی‌درنگ، معرفی می‌کنیم.

## ۲ - ۱ سامانه‌های بی‌درنگ

سامانه‌های بی‌درنگ نوعی از سامانه‌ها هستند که عملکرد صحیح آن‌ها، تنها به اجرای بدون خطای وظایف تعریف شده ختم نمی‌شود بلکه باید وظایف خود را در مهلت‌های زمانی مشخص شده برای آن‌ها تکمیل کنند. به عبارتی در این سامانه‌ها یک سری نیازمندی‌های زمانی وجود دارد که برآورده نکردن آن‌ها منجر به شکست سامانه می‌شود [۵]. سامانه‌های بی‌درنگ به سه نوع زیر تقسیم می‌شود:

- سامانه‌های بی‌درنگ نرم<sup>۱</sup>: سامانه‌هایی که انجام وظایف پیش از مهلت تعیین شده اهمیت دارد اما از دست دادن مهلت باعث مختل شدن عملکرد سامانه نمی‌شود؛ صرفاً ممکن است کیفیت خروجی‌های تولید شده را کاهش دهد.
- سامانه‌های بی‌درنگ سفت<sup>۲</sup>: این سامانه‌ها شبیه سامانه‌های نرم هستند، با این تفاوت که از دست دادن مهلت باعث از بین رفتن بهره‌ی سامانه می‌شود.
- سامانه‌های بی‌درنگ سخت<sup>۳</sup>: سامانه‌هایی که بسیار مهم است که زمان‌بندی وظایف به طور قطعی پیش از مهلت تعیین شده انجام شوند در غیر این صورت به خسارت‌های جانی یا مالی منجر می‌شود.

شکل ۱، نمودار بهره‌ی این سامانه‌ها را با هم مقایسه می‌کند. خط‌چین عمودی در این نمودارها، بیانگر نقطه‌ی شکست است.



شکل ۱ - انواع سیستم‌های بی‌درنگ

<sup>۱</sup> Soft  
<sup>۲</sup> Firm  
<sup>۳</sup> Hard

## ۲ - ۱ - ۱ وظایف سامانه‌های بی‌درنگ

یک کد که تعریف کننده‌ی بخشی از کارکردهای سامانه است را وظیفه<sup>۱</sup> می‌گویند. هر وظیفه می‌تواند چندین ورودی و خروجی داشته باشد. همچنین هر وظیفه می‌تواند توسط یک رخداد<sup>۲</sup> یا توسط فرارسیدن زمانی مشخص فعال شود. هر کار<sup>۳</sup> یک نمونه از وظیفه است که در سامانه منتشر شده است [۵]. مولفه‌های اصلی تعریف کننده‌ی یک وظیفه به شرح زیر هستند:

- T: تناوب یا دوره‌ی تناوب که حداقل فاصله‌ی زمانی بین دو نمونه‌ی کار از یک وظیفه است.
- D: مهلت زمانی نسبی یا ضرب العجل نسبی که مشخص می‌کند یک نمونه کار، تا چند واحد زمانی بعد از انتشار برای اجرا شدن فرصت دارد.
- C: زمان اجرا یا بدترین زمان اجرا (WCET) مشخص می‌کند که یک وظیفه برای تکمیل شدن، باید چند واحد زمانی توسط پردازنده اجرا شود.
- U: بهره‌وری یا استفاده که از رابطه‌ی  $U = \frac{C}{T}$  به دست می‌آید و میزان کار در واحد زمان یک وظیفه را مشخص می‌کند. به عبارت دیگر این مولفه مشخص می‌کند که یک وظیفه چه نسبتی از کل زمان خود را به اجرا شدن اختصاص می‌دهد. در پردازنده‌هایی که سرعت آن‌ها بیشتر از واحد و برابر 5 است، بهره‌وری به صورت  $U = \frac{C}{T \times S}$  محاسبه می‌شود.

بسته به نوع سامانه، ممکن است مولفه‌ی اولویت، که با نماد  $\rho$  یا P مشخص می‌شود، نیز برای هر وظیفه تعریف شود. هر چه مقدار عددی اولویت اختصاص داده شده به یک وظیفه بزرگ‌تر باشد، اولویت اجرای آن وظیفه پایین‌تر است. مولفه‌های دیگری نیز برای وظایف تعریف می‌شود که تنها بعد از انتشار یک نمونه از وظیفه و اجرای کامل آن، قابل محاسبه است. برای مثال زمان حقیقی اجرا که می‌تواند با بدترین زمان اجرای وظیفه متفاوت باشد و عموماً از آن کوچک‌تر است یا زمان پاسخگویی<sup>۴</sup>، که با نماد R مشخص می‌شود و نشان می‌دهد از لحظه‌ی انتشار وظیفه تا لحظه‌ای که اجرای آن تکمیل شده‌است، چند واحد زمانی گذشته است [۵].

<sup>1</sup> Task

<sup>2</sup> Event

<sup>3</sup> Job

<sup>4</sup> Response Time

در عموم مدل‌های تعریف وظیفه، رابطه‌ای میان دوره‌ی تناوب و مهلت نسبی وظایف وجود دارد. مهلت ضمنی<sup>۱</sup> حالتی است که مهلت نسبی و تناوب یک وظیفه با هم برابر باشند. مهلت محدود<sup>۲</sup> حالتی است که مهلت نسبی وظیفه از دوره‌ی تناوب آن کوچک‌تر باشد. مهلت دلخواه<sup>۳</sup> حالتی است که رابطه‌ی ثابت و مشخصی میان دوره‌ی تناوب و مهلت نسبی وجود نداشته باشد [۵].

وظایف با توجه به نحوه انتشار کارهایشان به دو دسته‌ی کلی متناوب و غیر متناوب تقسیم می‌شوند. وظایف متناوب با فواصلی که دقیقاً برابر با دوره‌ی تناوب آن‌ها است، منتشر می‌شوند و در یک بازه‌ی زمانی مشخص به صورت مرتب اجرا می‌شوند. وظایف غیرمتناوب نیز به دو دسته‌ی اسپورادیک<sup>۴</sup> و رخداد-محور تقسیم می‌شوند. وظایف اسپورادیک گرچه به صورت مرتب منتشر نمی‌شوند و نمی‌توان زمان فرارسیدن دقیق آن‌ها را مشخص کرد اما فاصله‌ی بین هر دو انتشار از آن‌ها، حداقل به اندازه‌ی دوره‌ی تناوب است. عموم وظایف بی‌درنگ نیز از نوع اسپورادیک هستند اما برای ساده‌سازی تحلیل‌ها در مطالعات، اکثراً از وظایف متناوب استفاده می‌شود [۱].

با در نظر گرفتن خطای زمانی یا خطای اجرایی به عنوان شکست، وظایف بحرانی می‌توانند به دو دسته‌ی کلی تقسیم شوند:

- بحرانی-ایمن: وظایفی که شکست در آن‌ها منجر به خسارات ایمنی و جانی می‌شود.
- بحرانی-عملیاتی<sup>۵</sup>: وظایفی که شکست در آن‌ها منجر به خسارات عملیاتی و مالی می‌شود.

## ۲ - ۱ - ۲ زمانبندی سامانه‌های بی‌درنگ

زمانبندی کردن به معنای اختصاص یک (یا چند) وظیفه به جفت (پردازنده‌ها)، بازه‌ی زمانی) برای اجرا شدن، می‌باشد. تعریف رسمی زمانبندی<sup>۶</sup> در تعریف ۲-۲ آورده شده‌است. وظیفه امکان‌پذیر<sup>۷</sup> در تعریف ۱-۲، زمانبندی امکان‌پذیر در تعریف ۲-۳، مجموعه وظایف امکان‌پذیر در تعریف ۴-۲ و زمانبندی‌پذیری<sup>۸</sup> در تعریف ۵-۲ به صورت رسمی تعریف شده‌اند.

<sup>۱</sup> Implicit Deadline

<sup>۲</sup> Constrained Deadline

<sup>۳</sup> Arbitrary Deadline

<sup>۴</sup> Sporadic

<sup>۵</sup> Mission-Critical

<sup>۶</sup> Schedule

<sup>۷</sup> Feasible

<sup>۸</sup> Schedulability

**تعريف ۱-۲:** در یک سامانه تک‌هسته‌ای با پردازنده‌ای با سرعت واحد، یک وظیفه امکان‌پذیر است اگر و تنها اگر  $D_i \leq T_i$  باشد [۵].

**تعريف ۲-۲:** برای یک مجموعه وظایف  $\{\tau_1, \tau_2, \dots, \tau_n\}$  زمان‌بندی یکتابع نگاشت به صورت  $\sigma: \mathbb{R}_+ \rightarrow \mathbb{N}$  است که به هر بازه‌ی زمانی  $(t_i, t_{i+1}]$  عدد طبیعی  $k$  را اختصاص می‌دهد که  $k=0$  به معنی بی‌کار بودن پردازنده در بازه‌ی زمانی مذکور و  $k > 0$  به معنی اجرای وظیفه‌ی  $\tau_k$  به روی پردازنده در بازه‌ی زمانی مذکور است [۵].

**تعريف ۳-۲:** یک زمان‌بندی امکان‌پذیر است، اگر و تنها، تمامی نیازمندی‌های تعریف شده برآ آن را برآورده سازد. نیازمندی تعریف شده به صورت عمومی «از دست ندادن هیچ مهلت زمانی» می‌باشد [۵].

**تعريف ۴-۲:** یک مجموعه وظایف امکان‌پذیر است، اگر و تنها اگر، یک زمان‌بندی امکان‌پذیر برای آن یافت شود [۵].

**تعريف ۵-۲:** مجموعه وظایف  $\tau$  با استفاده از الگوریتم A زمان‌بندی‌پذیر است، اگر و تنها اگر، الگوریتم A یک زمان‌بندی امکان‌پذیر برای مجموعه وظایف  $\tau$  تولید کند [۵].

برای آنکه بدانیم یک مجموعه وظایف با استفاده از الگوریتم زمان‌بندی A امکان‌پذیر است یا نه، یک راه بدیهی، اجرای الگوریتم زمان‌بندی بر روی مجموعه وظایف است اما پیش از شروع زمان‌بندی، به ازای هر الگوریتم، آزمون‌هایی وجود دارند که می‌توانند زمان‌بندی پذیر بودن یا نبودن یک مجموعه وظایف را مشخص کنند. این آزمون‌ها به سه دسته‌ی لازم، کافی و دقیق تقسیم می‌شوند [۵].

- آزمون لازم<sup>۱</sup>: رد شدن  $\tau$  در این آزمون نشان می‌دهد که  $\tau$  قطعاً زمان‌بندی‌پذیر نیست. برای مثال، بررسی آنکه بهره‌وری کل یک مجموعه وظایف از ۱ کوچک‌تر باشد برای زمان‌بندی RM<sup>۲</sup> یک آزمون لازم است.
- آزمون کافی<sup>۳</sup>: قبول شدن  $\tau$  در این آزمون نشان می‌دهد که  $\tau$  قطعاً زمان‌بندی‌پذیر است اما رد شدن به معنای عدم زمان‌بندی‌پذیری نیست و برای دانستن نتیجه‌ی قطعی، زمان‌بندی باید اجرا شود. برای مثال، آزمون پارک<sup>۴</sup> یک آزمون کافی برای الگوریتم DM<sup>۵</sup> است.

<sup>۱</sup> Necessary Test

<sup>۲</sup> Rate-Monotonic

<sup>۳</sup> Sufficient Test

<sup>۴</sup> Park's Test

<sup>۵</sup> Deadline-Monotonic

- آزمون دقیق<sup>۱</sup>: رد شدن و قبول شدن  $\tau$  در این آزمون، زمان‌بندی‌پذیر بودن یا نبودن  $\tau$  را به صورت قطعی مشخص می‌کند. برای مثال، تحلیل زمان پاسخگویی<sup>۲</sup> (RTA) یک آزمون دقیق برای الگوریتم‌های اولویت ثابت است.

الگوریتم‌های زمان‌بندی، از نظر تخصیص اولویت اجرا به وظایف، به سه دسته‌ی کلی اولویت ثابت وظیفه<sup>۳</sup> (FTP)، اولویت ثابت کار<sup>۴</sup> (FJP)، اولویت پویا<sup>۵</sup> (DP) تقسیم می‌شوند [۱]. الگوریتم زمان‌بندی بهینه<sup>۶</sup>، الگوریتمی است که اگر نتواند یک مجموعه وظایف را زمان‌بندی کند، هیچ الگوریتم دیگری نیز نمی‌تواند. در اینجا به سه نمونه از الگوریتم‌های اولویت‌دار پایه و بهینه زمان‌بندی در سامانه‌های بی‌درنگ اشاره می‌کنیم:

- الگوریتم DM: یک الگوریتم FTP است که اولویت وظایف را بر اساس مهلت نسبی آن‌ها مشخص می‌کند؛ هر چه مهلت نسبی یک وظیفه کوتاه‌تر، اولویت وظیفه بالاتر. در سامانه‌هایی با مهلت محدود، الگوریتم DM یک الگوریتم بهینه برای تخصیص اولویت به وظایف است.
- الگوریتم RM: یک الگوریتم FTP است که اولویت وظایف را بر اساس نرخ آن‌ها مشخص می‌کند؛ هر چه نرخ یک وظیفه بیشتر، اولویت آن وظیفه بالاتر. نرخ یک وظیفه نیز، معکوس تناوب آن است. الگوریتم RM یک الگوریتم بهینه برای تخصیص اولویت به وظایفی با مهلت ضمنی است.
- الگوریتم EDF: یک الگوریتم FJP است که در هر لحظه، کاری را که ضرب العجل آن نزدیک‌تر است برای اجرا انتخاب می‌کند. در میان الگوریتم‌های FJP، الگوریتم EDF یک الگوریتم بهینه است.

## ۲ - ۲ سامانه‌های بحرانی مختلط

سامانه‌های بی‌درنگ بحرانی، سامانه‌هایی سخت‌افزاری یا نرم‌افزاری شامل یک یا چند عملکرد بحرانی-ایمن هستند که بروز خطا در اجرای آن‌ها موجب به بار آمدن خسارات جانی یا مالی سنگین می‌شود. در این سامانه‌ها، تلاش برای اخذ گواهینامه‌های استاندارد ایجاب می‌کند که اجرای وظایف بحرانی-ایمن یا وظایفی که بر اساس تعاریف استانداردهایی مانند IEC 61508-DO-178B یا IEC 61508، از درجه‌ی بحرانیت بالایی برخوردار هستند، در مهلت تعیین

<sup>۱</sup> Exact Test

<sup>۲</sup> Response Time Analysis

<sup>۳</sup> Fixed Task Priority

<sup>۴</sup> Fixed Job Priority

<sup>۵</sup> Dynamic Priority

<sup>۶</sup> Optimal

شده‌ی اجرای آن‌ها، تضمین شود. هر چه درجه‌ی بحرانیت یک وظیفه بالاتر باشد، به درجه‌ی اطمینان بالاتری برای تضمین اجرا نیاز دارد. به یک نمونه از تعریف این درجه‌های بحرانیت در جدول ۱ اشاره شد.

برای برآورده ساختن این نیازمندی مهم، امروزه روش‌های مبتنی بر بخش‌بندی<sup>۱</sup> زمانی یا مکانی منابع سامانه، به طور گسترده‌ای توسط سامانه‌های بی‌درنگ بحرانی مورد استفاده قرار می‌گیرند. همین‌طور روش‌هایی مانند تامین منابعی فراتر از حد نیاز سامانه<sup>۲</sup> نیز مرسوم هستند. علت استفاده از این روش‌ها، جلوگیری از تداخل وظایف از درجه‌ی بحرانی پایین‌تر با وظایف از درجه‌ی بحرانی بالا می‌باشد تا اجرای بدون خطای وظایف بحرانی‌ایمن، تضمین شود. از طرفی این روش‌ها عموماً هزینه‌ی زیادی دارند و بهره‌وری از منابع سامانه در آن‌ها بسیار کم است. این هزینه‌ها تنها شامل هزینه‌های پردازشی یا مالی نیستند و زمان و انرژی را نیز شامل می‌شوند، که در سامانه‌های بی‌درنگ بحرانی با منابع انرژی محدود بسیار حائز اهمیت است. از این رو، تمایل صنعت برای استفاده از سامانه‌های بی‌درنگ بحرانی مجتمع‌سازی‌شده مانند IMA<sup>۳</sup> که وظایف از درجه‌ی بحرانیت متفاوت را بر روی یک بستر سخت‌افزاری اجرا می‌کنند، افزایش یافته است [۶].

مراجع صدور گواهی که وظیفه دارند از صحت اجرای وظایف بحرانی-ایمنی اطمینان یابند، برای کسب این اطمینان عموماً با روش‌هایی سخت‌گیرانه به محاسبه‌ی WCET وظایف می‌پردازند. این روش‌های سخت‌گیرانه منجر به تولید مقادیر WCET بسیار بزرگی می‌شوند که در اجرای واقعی سامانه به ندرت و بعض‌ا هیچ‌گاه اتفاق نمی‌افتد. این مساله آنجایی اهمیت می‌یابد که در سامانه‌های مجتمع‌سازی شده، وظایف از درجه‌ی بحرانی پایین‌تر نیز به دلیل احتمال تاثیرگذاری بر وظایف از درجه‌ی بحرانی بالا، با بالاترین حد سخت‌گیری مورد سنجش قرار می‌گیرند. در صورتی که هیچ تضمینی وجود ندارد که تعیین WCET سخت‌گیرانه برای نرم‌افزاری با کیفیت پایین‌تر لزوماً به اطمینان از اجرای صحیح آن منجر شود [۲].

این نکته حائز اهمیت است که اکثر وظایف سامانه‌های بی‌درنگ در دنیای واقعی، وظایفی اسپورادیک هستند که در دوره‌های مرتب منتشر نمی‌شوند اما برای تضمین زمان‌بندی آن‌ها، تصور می‌شود که وظایف به صورت مکرر در حال انتشار و استفاده از منابع سامانه هستند [۱]. با در نظر گرفتن این نکته در کنار توضیحاتی که ارائه شد، می‌توان دریافت که در سامانه‌های بی‌درنگ بحرانی، منابع سامانه در بیشتر مواقع بی‌کار<sup>۴</sup> هستند.

<sup>۱</sup> Partitioning

<sup>۲</sup> Over-Provisioning

<sup>۳</sup> Integrated Modular Avionics

<sup>۴</sup> Idle

سامانه‌های بحرانی مختلط، نوعی از سامانه‌های بی‌درنگ بحرانی مجتمع‌سازی‌شده هستند که علاوه بر اجرای وظایف از درجه‌های بحرانی گوناگون در یک بستر سخت‌افزاری مشترک، به وظایف اجازه می‌دهند که منابع سامانه را به اشتراک بگذارند و برای آن‌ها رقابت کنند. با این روش اشتراک منابع، هزینه‌ها کاهش و بهره‌وری سامانه افزایش می‌یابد. در عین حال، هدف اصلی آن‌ها، مانند هر سامانه بی‌درنگ بحرانی دیگر، اجرای صحیح وظایف بحرانی-ایمن یا وظایف از درجه‌ی بحرانی بالا در مهلت زمانی معین شده است.

پس، مساله‌ی اصلی سامانه‌های بحرانی مختلط، مساله‌ای از جنس ایجاد تعادل میان کارایی و اطمینان است. این سامانه‌ها تلاش می‌کنند تا در حین حفظ اطمینان از اجرای ایمن سامانه که در حد شدید با جداسازی<sup>۱</sup> وظایف از درجه‌ی بحرانی متفاوت محقق می‌شود، کارایی سامانه را نیز با مجتمع‌سازی وظایف و بهاشتراک‌گذاری منابع، افزایش دهند. بحرانیت-مختلط یک مفهوم است که می‌تواند در سطوح معماری، زمان‌بندی، تخصیص منابع و مدیریت خطا در سامانه اعمال شود [۲].

در زمینه‌ی زمان‌بندی سامانه‌های بحرانی مختلط، وستال در سال ۲۰۰۷ با تعریف مدلی جدید برای توصیف وظایف بحرانی مختلط که منطبق بر مجموعه وظایف ارائه شده در سوال ۱-۱ می‌باشد و ارائه‌ی یک روش تخصیص اولویت ثابت و تحلیل زمان‌بندی مبتنی بر زمان پاسخگویی برای این مدل از وظایف، دیدگاه جدیدی را در تعریف و زمان‌بندی سامانه‌های بحرانی مختلط پایه‌گذاری کرد. او بیان کرد که برای تعریف یک وظیفه در سامانه بحرانی مختلط، می‌توان از چندین مقدار WCET استفاده کرد و برای بررسی زمان‌بندی یک وظیفه از یک درجه‌ی بحرانیت خاص، باید از دیدگاه همان وظیفه به سامانه نگاه و آن را بررسی کرد [۲]. زمان‌بندی وستال، ما را به پاسخ ۱-۱ می‌رساند.

پس از وستال، باروآح و همکارانش در سال ۲۰۰۸ با تخصیص درجه‌ی بحرانیت به حالت عملیاتی سامانه، مدل بحرانیت انطباق پذیر، به طور مخفف AMC، را برای سامانه‌های بحرانی مختلط ارائه دادند [۶]. در این دیدگاه سامانه در حالت عملیاتی عادی شروع به فعالیت می‌کند و هر وظیفه با WCET خوش‌بینانه‌ای که طراح سامانه به آن اختصاص داده است، اجرا می‌شود. در صورتی که وظیفه‌ای از این زمان اجرای خوش‌بینانه تجاوز کند، که به اصطلاح به آن overrun می‌گویند، سامانه وارد حالت عملیاتی بحرانی می‌شود. در حالت عملیاتی بحرانی، تمامی وظایف از درجه‌ی بحرانی پایین حذف می‌شوند و هر وظیفه با WCET بدینانه‌ای که مرجع صدور گواهینامه به

<sup>۱</sup> Isolation

آن اختصاص داده است، اجرا می‌شود. تعریف رسمی این مدل اجرا به ازای بیش از ۲ درجه‌ی بحرانیت، در ادامه ارائه خواهد شد. چنین دیدگاهی ما را به پاسخ ۲-۱ می‌رساند.

پس از آن تحقیقات گستردۀ‌ای بر روی این مدل‌ها صورت گرفت و مدل‌های جدیدتر و جامع‌تری ارائه شد که در ادامه به آن‌ها می‌پردازیم اما در حال حاضر، تعریف عمومی سامانه‌های بحرانی مختلط، بر مبنای ترکیبی از همین دو دیدگاه ارائه می‌شود.

## ۲ - ۲ - ۱ مدل سامانه‌های بحرانی مختلط

یک سامانه بحرانی مختلط که دارای وظایفی از  $k$  درجه‌ی متفاوت بحرانیت است را در نظر بگیرید. مجموعه وظایف  $\tau$  در این سامانه تعریف شده و هر وظیفه از این مجموعه به صورت چهارتایی  $(T_i, D_i, \vec{C}_i, L_i, \varphi_i) = \tau_i$  تعریف می‌شود. هر کدام از این وظایف می‌توانند به طور بالقوه منجر به تولید مجموعه‌ای نامتناهی از کارها شوند. در جدول ۲، هر کدام از این مولفه‌ها توضیح داده شده است [۳].

جدول ۲- تعریف مولفه‌های وظیفه‌ی بحرانی مختلط

تعریف مولفه	مولفه
دوره یا تناوب یک وظیفه که مشخص می‌کند هر دو نمونه کار متوالی از یک وظیفه، از نظر زمان چقدر با هم فاصله دارند.	$T_i$
مهلت نسبی یک وظیفه که مشخص می‌کند هر نمونه کار از وظیفه، بعد از لحظه‌ی انتشار، چند واحد زمانی برای تکمیل اجرای خود فرست دارد.	$D_i$
یک بردار از بدترین زمان‌های اجرای یک وظیفه یا WCET های آن است که یک درجه‌ی بحرانی را به بدترین زمان اجرای وظیفه در آن درجه‌ی بحرانیت نگاشت می‌کند.	$\vec{C}_i$
یک عدد طبیعی کوچکتر یا مساوی $k$ است که درجه‌ی بحرانیت یک وظیفه را مشخص می‌کند.	$L_i$

به ازای هر وظیفه رابطه‌ی ۱-۲ برقرار است.

$$L_1 \leq L_2 \Rightarrow \vec{C}(L_1) \leq \vec{C}(L_2) \quad (1-2)$$

البته از آنجایی که تحقیقات و بررسی‌ها عموماً بر روی سامانه‌هایی با ۲ سطح بحرانیت که به صورت  $LO < HI$  تعریف می‌شوند، انجام می‌شود، مولفه‌ی برداری  $\vec{C}_i$  به دو مولفه‌ی  $C_i(HI)$  و  $C_i(LO)$  شکسته می‌شود و رابطه‌ی  $C_i(HI) \geq C_i(LO)$  برای آن برقرار است [۳]. مرسوم است که به  $C_i(LO)$  بدترین زمان اجرای خوش‌بینانه و به  $C_i(HI)$  بدترین زمان اجرای بدبینانه گفته شود.

در هر لحظه، یک مولفه‌ی  $\Gamma$  برای سامانه تعریف شده است که درجه‌ی بحرانیت حالت عملیاتی (رفتار) سامانه را مشخص می‌کند. عموماً در ابتدای شروع سامانه، این مقدار برابر ۱ یا  $L_0$ ، در اصل کمترین درجه‌ی بحرانیت ممکن، قرار داده می‌شود. عموماً به  $\Gamma = L_0$  حالت عملیاتی عادی و به  $\Gamma = HI$  حالت عملیاتی بحرانی می‌گویند [۳]. در مدل بحرانیت انطباق‌پذیر که حالت پیش‌فرض پیاده‌سازی سامانه‌های بحرانی مختلط می‌باشد، زمانی که سامانه به حالت عملیاتی از درجه‌ی بحرانی  $\Gamma$  وارد شود، کلیه‌ی وظایف از درجه‌ی بحرانی  $\Gamma < L_i$  از سامانه حذف می‌شوند و فقط وظایفی از درجه‌ی بحرانی  $\Gamma$  و بالاتر در سامانه باقی مانده و اجرا می‌شوند.

مفهوم اصلی در مدل‌سازی وستال از سامانه‌های بحرانی مختلط، ارائه‌ی مجموعه وظایفی بود که مولفه‌های تعریف‌کننده‌ی آن مستقل و ثابت نیست، بلکه به درجه‌ی بحرانی یا درجه‌ی اطمینان مورد نیاز برای آن وظیفه وابسته است. در مدل وستال، مولفه‌ی  $C$  وظایف به صورت وابسته تعریف می‌شود اما بعد از آن تحقیقاتی در زمینه‌ی وابسته‌سازی مولفه‌های  $D$  و  $T$  نیز انجام شده است [۳].

به طور کلی، مدل وظایف ارائه شده توسط وستال، یک مدل بسیار گویا<sup>۱</sup> و قدرمند است که با استفاده از آن می‌توان سامانه‌های سنتی و بحرانی مختلط بسیاری را توصیف کرد. برای مثال، در صورتی که بدترین زمان اجرای یک وظیفه از درجه‌ی بحرانیت  $L_0$  در سطح اجرایی  $HI$  مجھول باشد یا قابل محاسبه نباشد، می‌توان مقدار  $C(HI)$  آن وظیفه را برابر با  $\infty$  قرار داد و حتی در صورت وجود مجھولات نیز، مجموعه وظایف را مدل کرد [۱].

## ۲ - ۲ - چالش‌های پیاده‌سازی

در پیاده‌سازی مدل‌های بحرانی مختلط، دو چالش اصلی و مهم وجود دارد [۳]:

۱) جداسازی: وظایف از درجه بحرانی مجزا نباید تاثیری بر یکدیگر داشته باشند. علی الخصوص بروز خطا در وظایف از درجه‌ی بحرانی پایین‌تر، نباید بر اجرای وظایف از درجه‌ی بحرانی بالاتر و به طور کلی هیچ وظیفه‌ی دیگری تاثیر بگذارد.

۲) رعایت دوره‌ی تناوب: وظایف نباید کارهایی منتشر کنند که فاصله‌ی بین آن‌ها از دوره‌ی تناوب کوچک‌تر است.

<sup>۱</sup> Expressive

۳) رعایت بدترین زمان اجرا: در هر سطح عملیاتی سامانه، هیچ وظیفه‌ای نباید از بدترین زمان اجرای مطابق با درجه‌ی بحرانیت سامانه خود تجاوز کند.

۴) تخصیص منابع: در سامانه‌هایی که منابع مشترکی غیر از پردازنده دارند، اولویت دسترسی وظایف از درجه‌های بحرانی مختلف به منابع سامانه باید کنترل شود به طوری که هیچ وظیفه‌ای از درجه‌ی بحرانی پایین‌تر، اجرای وظیفه از درجه‌ی بحرانی بالاتر را مسدود نکند. همینطور بروز خطا در حین دسترسی به یک منبع، نباید به وظایف دیگر نفوذ کند. از این نظر، پیاده‌سازی سامانه‌های بحرانی مختلط می‌تواند با چالش‌های سخت‌افزاری نیز همراه باشد.

## ۲ - ۲ - ۳ چالش‌های صنعتی‌سازی

در صنعتی‌سازی سامانه‌های بحرانی مختلط، دو دغدغه‌ی اساسی توسط طراحان سامانه مطرح می‌شود [۳]:

۱) در حالت عملیاتی بحرانی، وظایف از درجه‌ی بحرانی LO نباید به طور کلی حذف شوند. چرا که وجود و اجرای این وظایف برای تضمین عملکرد سامانه ضروری است. درست است که این وظایف بحرانیت ایمنی ندارند اما برخی از آن‌ها بحرانیت عملیاتی دارند و اجرای سامانه بدون آن‌ها معنی ندارد.

۲) سامانه‌هایی که برای مدت طولانی کار می‌کنند، باید سازوکاری برای بازگشت به حالت عملیاتی عادی داشته باشند.

محققان استدلال می‌کنند که در دنیای واقعی، انتظار نمی‌رود که سامانه وارد حالت عملیاتی بحرانی شود. حالت عملیاتی بحرانی یک حالت بسیار بدینانه از اجرای سامانه است که تعریف آن صرفا برای تضمین زمان‌بندی سامانه مطابق استاندارهای ایمنی است. چنین حالت بدینانه‌ای، تنها برای بررسی‌های تئوری تعریف شده است و در عمل، یک سامانه بحرانی مختلط به احتمال خیلی زیادی در تمام طول اجرای خود در حالت عملیاتی عادی باقی می‌ماند [۳].

البته در سامانه‌هایی که به طور کلی بحرانیت کمتری دارند و استاندارهای سهل‌انگارانه‌تری برای آن‌ها نیاز است، برای مثال سامانه‌های بحرانی مختلطی که وظایف بحرانی‌ایمن ندارند، احتمال بروز خطا در وظایف بیشتر است و نتیجتاً احتمال دارد که در یک اجرای واقعی، سامانه وارد حالت بحرانی بالا شود. در چنین حالتی، دغدغه‌ی طراحان سامانه به‌جا است و منطقی نیست که در حالت عملیاتی بحرانی، همه‌ی وظایفی که از بحران پایین هستند حذف شوند یا آنکه سامانه در تمام طول اجرا در حالت عملیاتی بحرانی باقی بماند [۳].

لازم به ذکر است که در برخی از سامانه‌های چند بحرانیتی، یک وظیفه زمانی به بحران بالا تعلق دارد که اجرای آن برای عملکرد صحیح سامانه ضروری باشد و محقق شدن هدف سامانه، منوط به اجرای این وظایف است. در

چنین حالتی، وظایفی که از بحران پایین هستند، وظایفی کم‌اهمیت و ناچیز هستند که روی عملکرد سامانه تاثیری ندارند و تنها برای برآورده کردن برخی جنبه‌های کیفی به سامانه اضافه شده‌اند [۳].

در همین‌جا می‌توان به این چالش پی برد که در یک سامانه بحرانی مختلط که هدف اصلی آن اجرای وظایف از بحرانیت‌های متفاوت بر روی منابع مشترک است، بحرانیت تعریف واحدی ندارد و بحرانی بودن یک وظیفه، لزوماً به معنای بحرانی‌ایمن بودن یک وظیفه نیست. بسته به نگاه طراح سامانه، یک وظیفه‌ی غیر-بحرانی-ایمن می‌تواند بحران بالایی داشته باشد. فی الواقع تصویر استاندارها از یک وظیفه‌ی بحرانی، وظیفه‌ای است که تحمل خطای بسیار پایینی دارد و برای مثال نمی‌تواند از حالت خطا خارج شود. یکی از چالش‌های اصلی صنعتی‌سازی سامانه‌های بحرانی مختلط، عدم وجود تعریفی دقیق برای مفهوم بحرانیت است [۳].

در سال ۲۰۱۷، در کنفرانسی در زمینه‌ی سامانه‌های بحرانی مختلط، مطرح شد که هدف مدل وستال، علاوه بر تضمین قابلیت اعتماد سامانه برای دریافت گواهینامه‌های استاندار، حفظ عملکردهای اصلی سامانه می‌باشد. در اصل، وستال مدلش را ارائه کرد تا عملکردهای اصلی یک سامانه، قربانی سیاست‌های تایید صلاحیت نشوند و هر دوی این اهداف در یک سامانه برآورده شوند. مدل وستال تاکیدی بر حذف هیچ وظیفه‌ای نداشت و تنها دیدگاه‌هایی متفاوت برای بررسی سامانه ارائه می‌کرد. تاکید مدل وستال، اتفاقاً بر کاهش نظاممند سطح کیفی سامانه است. مدل‌هایی که برای سامانه حالتی عملیاتی تعریف می‌کنند و وظایف از درجه‌ی بحرانی پایین را حذف می‌کنند (مثل مدل بحرانیت انطباق‌پذیر) بعدها برای افزایش توانایی زمان‌بندی مجموعه وظیفه‌هایی متنوع‌تر که مدل وستال قادر به زمان‌بندی آن‌ها نبود، معرفی شدند. در این کنفرانس، تحقیقاتی که بر این مدل اجرایی بنا شده‌اند، از نظر قابلیت صنعتی‌سازی، مردود اعلام می‌شوند. تحقیقات و الگوریتم‌های بسیار زیادی نیز مطرح شده‌اند که با رخداد خطای زمانی، به جای حذف همه‌ی وظایف از درجه‌ی بحرانی پایین‌تر، تنها برخی از وظایف را حذف می‌کنند یا فرکانس انتشار وظایف از درجه‌ی بحرانی پایین‌تر را کاهش می‌دهند و به طور کلی سازوکار ساختارمندتری برای افزایش درجه‌ی بحرانیت عملیات سامانه دارند [۳].

یکی دیگر از چالش‌های صنعتی‌سازی سامانه‌های بحرانی مختلط آن است که در سامانه‌ای که برای مثال، به تعریف ۷ سطح اولویت نیاز دارد، لزوماً ۷ مقدار برای بدترین زمان اجرای وظایف آن سامانه وجود ندارد.

### ۳ - زمانبندی سامانه‌های بحرانی مختلط

در این فصل، با تمرکز عمدۀ بر مجموعه‌ای از وظایف اسپورادیک، به بررسی جزئیات و چالش‌های مربوط به زمانبندی سامانه‌های بحرانی مختلط می‌پردازیم و برخی از الگوریتم‌های مطرح در این زمینه را مورد بررسی قرار می‌دهیم. توجه داشته باشید که توضیح مدل وظایف ارائه شده در این فصل و معنی واژگان مختصر، پیش‌تر در فصل ۲ - آمده است.

### ۳ - ۱ چالش‌های زمانبندی سامانه‌های بحرانی مختلط

پیش از هر چیز لازم است چند چالش عمدۀ در زمینه‌ی زمانبندی سامانه‌های بحرانی مختلط را به شمار بیاوریم و با در نظر داشتن این موارد به تحلیل الگوریتم‌های ارائه شده در این فصل بپردازیم:

- یک سامانه بحرانی مختلط ممکن است در هر یک از سطوح تعریف شده برای آن قابل زمانبندی باشد. اما، در صورتی که سامانه قابلیت تغییر سطح بحرانیت را داشته باشد، ممکن است در زمان انتقال از سطح بحرانی پایین‌تر به سطح بالاتر، قابل زمانبندی نباشد. به همین دلیل، تحلیل‌های ارائه شده برای بررسی زمانبندی این سامانه‌ها باید تمام حالت‌های ممکن برای تغییر سطح سامانه را در نظر بگیرند و ثابت کنند که سامانه در آن حالت‌ها نیز قابل زمانبندی است.
- مسئله زمانبندی سامانه‌های بحرانی مختلط یک مسئله NP-Hard است، حتی اگر سامانه تنها دارای دو سطح بحرانی باشد. به همین دلیل، تحلیل‌های ارائه شده برای زمانبندی پذیری این سامانه‌ها همگی دارای شروط کافی هستند و تاکنون هیچ تحلیل دقیقی ارائه نشده است [۳].
- بسیاری از الگوریتم‌های زمانبندی برای سامانه‌های بحرانی مختلط به یک سازوکار نظارت در زمان اجرا<sup>۱</sup> نیاز دارند. با این حال، تحقیقات اندکی بر روی نحوه پیاده‌سازی این سازوکارها صورت گرفته است. علاوه بر این، معمولاً در محاسبات، از سربار<sup>۲</sup> ناشی از این سازوکارها چشم‌پوشی می‌شود. در تحقیقی که توسط سیگریست<sup>۳</sup> و همکاران انجام شده است، آن‌ها نشان دادند که این سازوکارها می‌توانند سرباری ۹۷ درصدی داشته باشند. آن‌ها توصیه می‌کنند که تمام مدل‌های زمانبندی باید گسترش یابند تا شامل مولفه‌هایی باشند که تأثیر سربار در زمان اجرا را در نظر بگیرند [۳].

<sup>۱</sup> Runtime Monitoring

<sup>۲</sup> Overhead

<sup>۳</sup> Sigrist

## ۳ - ۲ - زمان‌بندی اولویت ثابت با روش‌های مبتنی بر تحلیل زمان پاسخگویی

الگوریتم‌های اولویت ثابت قبصه‌ای<sup>۱</sup>، یکی از دسته‌های محبوب الگوریتم‌های زمان‌بندی در سامانه‌های بی‌درنگ هستند. بنابراین، تحقیقات اولیه در حوزه زمان‌بندی سامانه‌های بحرانی مختلط با مطالعه الگوریتم‌های اولویت ثابت آغاز شده و آن‌ها را برای مدل‌سازی وظایف بحرانی مختلط توسعه داده‌اند. در بیشتر این پژوهش‌ها، مسئله تخصیص اولویت و مسئله تحلیل زمان‌بندی مجموعه وظایف با استفاده از اولویت‌های تعیین‌شده بررسی می‌شوند. در مسئله تخصیص اولویت، محققان به دنبال الگوریتمی هستند که بر اساس منطقی ثابت به هر وظیفه اولویتی اختصاص دهد. در مسئله تحلیل زمان‌بندی نیز با یک مدل تغییر یافته از تحلیل‌های زمان‌بندی بی‌درنگ، برای الگوریتم خود آزمون زمان‌بندی طراحی می‌کنند.

در این بخش، به بررسی رشد و تکامل الگوریتم‌های اولویت ثابت برای سامانه‌های بحرانی مختلط می‌پردازیم که با استفاده از روش‌های مبتنی بر تحلیل زمان پاسخگویی تعریف شده‌اند. روش‌های دیگری از جمله زمان‌بندی زمان اضافی<sup>۲</sup> و تبدیل تناوب<sup>۳</sup> نیز وجود دارند که در این دسته قرار می‌گیرند اما در این گزارش به آن‌ها پرداخته نمی‌شود.

در این روش‌ها، از تحلیل زمان پاسخگویی، که به اختصار RTA نامیده می‌شود، برای تخصیص اولویت و همچنین طراحی آزمون‌های زمان‌بندی استفاده می‌شود. یکی از دلایل گسترده استفاده محققان از تحلیل‌های مبتنی بر زمان پاسخگویی آن است که در حوزه زمان‌بندی سامانه‌های بی‌درنگ سنتی با الگوریتم‌های زمان‌بندی اولویت ثابت، این تحلیل یک آزمون زمان‌بندی دقیق را ارائه می‌دهد [۵]. از این رو بدیهی است که این روش‌ها برای سامانه‌های بحرانی مختلط نیز توسعه داده شوند.

## ۳ - ۲ - ۱ - الگوریتم CrMPO<sup>۴</sup>

الگوریتم CrMPO یکی از ساده‌ترین و بدیهی‌ترین الگوریتم‌های تخصیص اولویت به وظایف بحرانی مختلط است. این الگوریتم جزو دسته‌ی الگوریتم‌های CM<sup>۵</sup> یا PC<sup>۶</sup> قرار می‌گیرد. در الگوریتم‌های CM، اولویت‌ها بر اساس سطح بحرانیت هر وظیفه اختصاص داده می‌شوند، به طوری که وظایف با بحرانیت بالاتر همواره اولویتی بالاتر از وظایف

<sup>۱</sup> Preemptive Fixed Priority

<sup>۲</sup> Slack Scheduling

<sup>۳</sup> Period Transformation

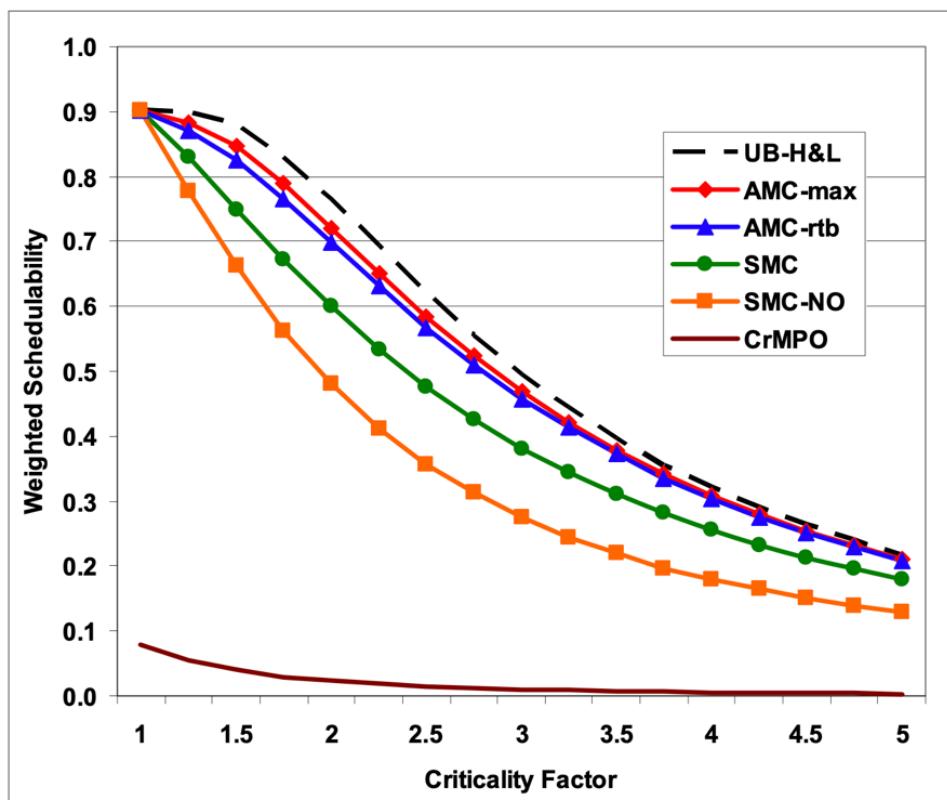
<sup>۴</sup> Criticality Monotonic Priority Ordering

<sup>۵</sup> Criticality Monotonic

<sup>۶</sup> Partitioned Criticality

با بحرانیت پایین‌تر خواهد داشت. در تخصیص اولویت CrMPO، وظایف ابتدا بر اساس درجهٔ بحرانیت مرتب می‌شوند و در صورت یکسان بودن درجهٔ بحرانیت، مرتب‌سازی بر اساس الگوریتم DM انجام می‌گیرد.

مزیت این نوع الگوریتم‌ها در آن است که بروز خطای زمانی در وظایف با درجهٔ بحرانیت پایین‌تر، موجب انتقال خطای وظایف با درجهٔ بحرانیت بالاتر نمی‌شود، چرا که این وظایف همواره قبل از وظایف با درجهٔ بحرانیت پایین‌تر اجرا شده‌اند. از این منظر، اینگونه الگوریتم‌ها دغدغه‌ی جداسازی را به خوبی در یک سامانهٔ بحرانی مختلط برآورده می‌کنند. علاوه بر آن، الگوریتم‌های CM نیازی به نظارت در زمان اجرا ندارند. اما همان‌طور که در شکل ۲ نشان داده شده است، این الگوریتم‌ها توانایی بسیار پایینی در زمان‌بندی مجموعه‌های وظایف بحرانی دارند، به این معنا که تعداد بسیار کمی از مجموعه‌های وظایف بحرانی را می‌توانند زمان‌بندی کنند [۶].



شکل ۲ مقایسه‌ی انواع الگوریتم‌های زمان‌بندی اولویت ثابت در زمان‌بندی سامانه‌های بحرانی مختلط [۶]

### ۳ - ۲ - ۲ - تحلیل چند‌بحرانیتی وستال و ناکارآمدی الگوریتم DM

در سامانه‌های بی‌درنگ سنتی که مهلت انجام وظیفه برابر یا کمتر از تناوب آن وظیفه است، الگوریتم DM به عنوان یک الگوریتم بهینه برای تخصیص اولویت به وظایف شناخته می‌شود. وستال در اولین مقاله خود پس از

ارائه مدل سامانه‌های بحرانی مختلط، اثبات می‌کند که الگوریتم DM برای زمانبندی مجموعه وظایف بحرانی مختلط، بهینه نیست. به این منظور، مجموعه وظایف نمایش داده شده در جدول ۳ را به عنوان نمونه ارائه می‌دهد .[۲]

جدول ۳ نمونه مجموعه وظایف بحرانی مختلط ارائه شده توسط وستال برای اثبات ناکارآمدی الگوریتم DM [۲]

i	D	T	L	C(LO)	C(HI)
1	2	2	B (LO)	1	2
2	4	4	A (HI)	1	1

در این نمونه، اگر به وظیفه ۱ بالاترین اولویت اختصاص یابد، آنگاه وظیفه ۲، پردازنده‌ای را می‌بیند که قبل از ۱۰۰٪ از زمان موجود سطح A خود را برای وظیفه ۱ کنار گذاشته است. با این حال، اگر به وظیفه ۲ بالاترین اولویت اختصاص داده شود طبق تحلیل چند-بحرانیتی رابطه ۱-۳، سامانه قابل اجرا است [۲].

$$R_i = \sum_{j:\rho_j \leq \rho_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i) \quad (1-3)$$

رابطه‌ی ۱-۳ معادله‌ای تغییر یافته از الگوریتم محاسبه‌ی بدترین زمان پاسخگویی جوزف-پاندیا<sup>۱</sup> است که وستال آن را برای بررسی زمانبندی سامانه‌های بحرانی مختلط ارائه داد. این رابطه یک رابطه‌ی بازگشتی است که مقدار  $R_i$  در آن از محاسبه‌ی بازگشتی سمت راست به روزرسانی می‌شود. محاسبه از  $R_i = C_j(L_i)$  آغاز می‌شود و تا زمانی که  $R_i > D_i$  یا  $R_i$  دیگر تغییر نکند ادامه می‌یابد. در حالت اول، مجموعه وظایف با اولویت‌های  $\rho$  قابل زمانبندی نیست [۲]. رابطه‌ی ۱-۳ در اصل صورتی تغییر یافته از RTA است که وستال برای تحلیل زمانبندی سامانه‌های بحرانی مختلط ارائه داد. طبق این رابطه، زمانبندی‌پذیری هر وظیفه با توجه به سطح بحرانی همان وظیفه سنجیده می‌شود.

وستال مدل اختصاص اولویت آذلی<sup>۲</sup> را نیز با به کارگیری رابطه‌ی ۱-۳ برای مجموعه وظایف بحرانی مختلط تعمیم داد. در الگوریتم تخصیص اولویت وستال، ابتدا هیچ کدام از وظایف اولویتی ندارند و با به کار گرفتن رابطه‌ی ۱-۳، وظایف به ترتیب از پایین‌ترین اولویت به بالاترین اولویت دسته‌بندی می‌شوند. در اختصاص اولویت به وظیفه‌ی

<sup>۱</sup> Joseph-Pandya<sup>۲</sup> Audsley

ن، تصور می‌شود که تمامی وظایف دیگر که هنوز اولویت‌بندی نشده‌اند، اولویت بالاتری از  $\alpha$  دارند. در این الگوریتم به هر وظیفه یک اولویت یکتا اختصاص داده می‌شود و اولویت هیچ دو وظیفه‌ای با هم برابر نیست. در صورتی که چند وظیفه با درجه‌ی بحرانی متفاوت، با درجه‌ی اولویت یکسانی قابل زمان‌بندی باشند، وظیفه با درجه‌ی بحرانی بالاتر، اولویت بالاتر را به خود اختصاص می‌دهد [۲].

الگوریتم تخصیص اولویت وستال گرچه به نسبت الگوریتم‌هایی که بعد از آن ابداع شدند بدینانه‌تر است اما، امکان واژگونی بحرانیت<sup>۱</sup> را در شرایطی محاسبه‌شده فراهم می‌آورد. واژگونی بحرانیت به این معناست که وظیفه‌ای با درجه‌ی بحرانی پایین‌تر از وظیفه‌ای با درجه‌ی بحرانی بالاتر، اولویت بالاتری داشته باشد [۶]. بدیهی است که چنین تعبیری، به الگوریتم وستال اجازه می‌دهد تا توانایی زمان‌بندی تعداد وسیع‌تری از مجموعه وظایف را داشته باشد.

در پژوهش‌هایی که باروآح و همکارانش بعد از آن با اعمال تحلیل زمان پاسخگویی بر انواع الگوریتم‌های اولویت ثابت انجام دادند، نشان داده شد که الگوریتم وستال با نام SMC-NO<sup>۲</sup> در زمان‌بندی مجموعه وظایف بحرانی مختلط از الگوریتم CrMPO بسیار توانمندتر است. نمونه‌ای از نتایج این تحلیل‌ها در شکل ۲ آورده شده‌است [۶].

### ۳ - ۲ - ۳ روش‌های AMC و درجه‌ی بحرانیت یک رفتار

باروآح و همکارانش در سال ۲۰۱۱ با بهره‌گیری از نظرارت در زمان اجرا و تعریف مفهوم درجه‌ی بحرانیت یک رفتار<sup>۳</sup> (که در اصل تعبیری از درجه‌ی بحرانیت حالت عملیاتی سامانه است)، روش بحرانیت مختلط انطباق‌پذیر، به اختصار AMC، را ابداع کردند. این روش به صورت گسترده‌ای مورد تحقیق و بررسی قرار گرفت و بعد از آن تا حدودی به بخشی جدایی‌ناپذیر از تعریف سامانه‌های بحرانی مختلط تبدیل شد. آن‌ها دسته‌بندی جدول ۳ را برای الگوریتم‌های اولویت ثابت ارائه دادند:

جدول ۳ - دسته‌بندی انواع سامانه‌های بحرانی مختلط اولویت ثابت [۶]

نام روش	توضیح
بحرانیت بخش‌بندی شده (PC)	یک طرح استاندارد که گاهی اوقات به نام تخصیص اولویت مونوتونیک بحرانیت (CM) نامیده می‌شود. در این روش به هر

<sup>۱</sup> Criticality Inversion

<sup>۲</sup> Static Mixed Criticality – No Overrun

<sup>۳</sup> Criticality Level of a Behavior

وظیفه اولویتی متناظر با درجه بحرانیت آن وظیفه اختصاص داده می‌شود.	
طرحی منطبق بر مدل وستال که در آن هر کدام از کارهای مربوط به یک وظیفه می‌تواند تنها به اندازه‌ی بدترین زمان اجرای منسوب به درجه بحرانیت وظیفه ( $C_i(L_i)$ ) اجرا شود و بعد از آن بسته به مدل سامانه، آن کار متوقف شده یا برای بازیابی خطا در زمان دیگری زمان‌بندی می‌شود.	بحرانیت مختلط ثابت (SMC)
طرح نوآورانه‌ی باروآح و همکارانش که در آن در صورتی که هر کدام از کارهای مربوط به یک وظیفه از بدترین زمان اجرای منسوب به درجه بحرانیت وظیفه ( $C_i(L_i)$ ) تجاوز کنند، تمامی کارهای تمامی وظایفی که درجه بحرانیتی هماندازه با $L_i$ یا کمتر از آن دارند، متوقف شده و دیگر زمان‌بندی نمی‌شوند.	بحرانیت مختلط انطباق‌پذیر (AMC)

لازم به ذکر است که در تحقیقاتی که در سال‌های بعد انجام شد، دسته‌ی دیگری تحت عنوان Task-Level معرفی شد که بین SMC و AMC قرار دارد. در این مدل، در صورت تجاوز یک کار از درجه‌ی  $L_i$ ، تمامی کارهای وظیفه‌ی  $i$  را از سیستم حذف می‌کند و باقی کارهایی که از درجه‌ی  $L_i$  هستند را در سیستم باقی می‌گذارد [۳].

باروآح و همکارانش با دلایل مطرح شده در بخش ۳ - ۲ - ۱ بیان کردند که الگوریتم‌های PC برای زمان‌بندی سامانه‌های بحرانی مختلط قدرت کافی ندارند. سپس با این استدلال که به کارگیری روش‌های نظارت در زمان اجرا در سامانه‌های بی‌درنگ روشی مرسوم است و در صنعت نیز مورد استفاده قرار می‌گیرد به ابداع روش نوآورانه‌ی AMC پرداختند. افزودن سازوکارهای نظارت در زمان اجرا و اعمال بودجه<sup>۱</sup>، به کارهای وظایف از درجه‌های بحرانیت متفاوت اجازه می‌دهد که اولویت‌هایی ترکیب شده داشته باشند و این قدرت زمان‌بندی الگوریتم‌ها را افزایش می‌دهد [۶].

بعد از آن به تحلیل روش‌های بحرانیت مختلط ثابت می‌پردازند. روش پیشنهاد شده توسط وستال که در بخش ۳ - ۲ - ۲ بیان شد با نام SMC-NO بررسی می‌شود و نسخه‌ای از آن که به همراه نظارت‌های زمان اجرا کار می‌کند را SMC نامیدند. تحلیل‌های زمان پاسخگویی این دو الگوریتم در جدول ۴ آورده شده‌است. همانطور که مشاهده می‌کنید SMC با به کارگیری نظارت‌های زمان اجرا می‌تواند با تبدیل  $C_j$  به  $C_j(\min(L_i, L_j))$  بالاترین حد<sup>۲</sup>

<sup>۱</sup> Budget Enforcement

<sup>۲</sup> Upperbound

دقیق‌تری برای زمان پاسخگویی و در نتیجه زمان‌بندی‌پذیری بیشتری را فراهم کند. برای محاسبه‌ی تحلیل زمان پاسخگویی SMC، هر سه حالت  $L_i < L_j$ ،  $L_i = L_j$  و  $L_i > L_j$  بررسی شده‌اند [۶].

جدول ۴ - تحلیل زمان پاسخگویی SMC-NO و SMC

SMC	SMC-NO
$R_i = C_i + \sum_{\tau_j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\min(L_i, L_j))$	$R_i = C_i + \sum_{\tau_j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\min(L_i, L_j))$
* (hp(i) نشان‌دهنده‌ی تمام وظایفی است که اولویتی بالاتر از وظیفه‌ی $\tau$ دارد)	

تعريف ۳-۱: در اجراهای مختلف، هر مجموعه وظیفه‌ای به طور کلی رفتارهای مختلفی را نشان می‌دهد: ممکن است وظایف مختلف در لحظات زمانی مختلف آغاز شوند و زمان اجرای واقعی متفاوتی داشته باشند. بیایید درجه بحرانیت یک رفتار را به عنوان کمترین سطح بحرانیت تعریف کنیم به طوری که هیچ وظیفه‌ای بیش از مقدار  $C$  خودش در این سطح بحرانیت، اجرا نشود. برای هر سطح بحرانیت  $L$ ، تمام وظایف با بحرانیت  $\leq L$  در هر رفتار با سطح بحرانیت  $L$  تا مهلتهای مقرر شان تکمیل خواهند شد [۶].

با بهره‌گیری از تعريف ۳-۱، الگوریتم AMC برای یک سامانه دو بحرانیتی<sup>۱</sup> با بحرانیت‌های  $LO > HI > LO$  در ۵ قدم تعريف می‌شود [۶]:

- قدم ۱: یک نشانگر سطح بحرانیت  $\Gamma$  وجود دارد که به  $LO$  مقداردهی اولیه می‌شود.
- قدم ۲: در حالی که  $\Gamma \equiv LO$ ، در هر لحظه وظیفه‌ای که بالاترین اولویت را دارد برای اجرا انتخاب می‌شود.
- قدم ۳: اگر وظیفه‌ای که در حال اجرا است برای زمان اجرای بحرانیت پایین ( $LO$ ) خود بدون نشان دادن اتمام کار، اجرا شود، آنگاه  $HI \leftarrow \Gamma$ .

<sup>۱</sup> Dual Criticality

- قدم ۴: هنگامی که  $\Gamma \equiv HI$  اجرا نخواهد شد. بنابراین، از این پس، در هر لحظه وظیفه‌ای که توسط وظیفه با بحرانیت بالا (HI) و بالاترین اولویت تولید می‌شود برای اجرا انتخاب می‌شود.
- قدم ۵: یک قاعده اضافی می‌تواند شرایطی را مشخص کند که در آن  $\Gamma$  به LO بازنشانی می‌شود. این امر می‌تواند به عنوان مثال، زمانی رخ دهد که هیچ وظیفه‌ای با بحرانیت بالا (HI) در یک لحظه خاص فعال نباشد.

البته این الگوریتم در ادامه مورد بررسی بیشتر و بهبود قرار گرفت. به عنوان مثال، از دو سطح بحرانیت به k سطح بحرانیت تعمیم داده شد. همچنین برای سناریوهای مختلفی از متوقف کردن وظایف با بحرانیت پایین، قبل یا بعد از تکمیل اجرای آن‌ها، بررسی‌هایی صورت گرفت که بحث در مورد آن‌ها از حوصله این گزارش خارج است [۳]. تخصیص اولویت‌ها به وظایف در این الگوریتم نیز با روش آذلی انجام می‌شود. تحلیل زمانبندی سامانه در حالتی که  $\Gamma \equiv LO$  است طبق رابطه‌ی ۲-۳ و در حالتی که سامانه در حالت  $\Gamma \equiv HI$  است طبق رابطه‌ی ۳-۳ انجام می‌شود. در رابطه‌ی ۳-۳ منظور از  $hp(i)$  وظایفی با اولویت بالاتر از وظیفه‌ی i هستند که  $HI = L_i$  دارند [۶].

$$R_i^{LO} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j^{LO}}{T_j} \right\rceil C_j(LO) \quad (2-3)$$

$$R_i^{HI} = C_i + \sum_{j \in hpH(i)} \left\lceil \frac{R_j^{HI}}{T_j} \right\rceil C_j(HI) \quad (3-3)$$

همانطوری که در بخش ۳ - ۱ اشاره کردیم، مساله‌ی زمانبندی پذیر بودن مجموعه وظایف بحرانی مختلط در حالت تبدیل از سطح بحرانی پایین به سطح بحرانی بالا (قدم ۳ از الگوریتم AMC)، یکی از چالش‌های اصلی زمانبندی سامانه‌های بحرانی مختلط است. تحلیل زمانبندی سامانه در حین این تغییر، به دو روش محاسبه می‌شود [۶]:

### ۳ - ۲ - ۱ شرط کافی<sup>۱</sup> AMC-rtb<sup>۱</sup>

در این تحلیل که یک تحلیل مبتنی بر زمان پاسخگویی می‌باشد، ابتدا رابطه‌ی SMC که در جدول ۴ آمده است به صورت رابطه‌ی ۴-۳ شکسته می‌شود. در رابطه‌ی ۴-۳ منظور از  $hpL(i)$  وظایفی با اولویت بالاتر از وظیفه‌ی  $i$  هستند که  $L_i = LO$  دارند [۶]:

$$R_i = C_i + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\min(L_i, L_j)) + \sum_{\tau_k \in hpL(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k(LO) \quad (4-3)$$

از آنجایی که در حالت  $HI \equiv \Gamma$  تمامی وظایف با بدترین زمان اجرای بدینانه خود تحلیل می‌شوند و هدف بررسی فقط تسک‌هایی با بحرانیت  $HI$  است، رابطه‌ی ۴-۳ به صورت ۵-۳ تغییر می‌کند [۶]:

$$R_i = C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(HI) + \sum_{\tau_k \in hpL(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k(LO) \quad (5-3)$$

رابطه‌ی ۵-۳ که در اصل مدلی تغییر کرده از تحلیل SMC است برای AMC بسیار سخت‌گیرانه است چون متوقف شدن وظایف با درجه بحرانیت  $LO$  را در نظر نمی‌گیرد. تبدیل حالت رفتار سامانه از  $LO$  به  $HI$  حتماً باید قبل از  $R_i^{LO}$  که در رابطه‌ی ۲-۳ محاسبه شد اتفاق بیافتد در غیر این صورت وظیفه‌ی  $i$  تحت تاثیر تبدیل حالت سامانه قرار نمی‌گیرد و پیش از آن به اتمام رسیده است. از این رو حالت دقیق‌تر رابطه‌ی ۳-۵ در رابطه‌ی ۳-۶ نمایش داده شده است. رابطه‌ی ۳-۶ همان تحلیل زمان‌بندی مبتنی بر زمان پاسخگویی AMC-rtb می‌باشد. از آنجایی که این رابطه با به احتساب آوردن  $R_i^{LO}$  به دست آمده است به آن حد زمان پاسخگویی (rtb) گفته می‌شود و  $R_i^*$  مقدار بدترین زمان پاسخگویی وظیفه‌ی  $i$  در حالت تبدیل سامانه را به دست می‌دهد [۶].

$$R_i^* = C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j(HI) + \sum_{\tau_k \in hpL(i)} \left\lceil \frac{R_i^{LO}}{T_k} \right\rceil C_k(LO) \quad (6-3)$$

از نظر شهودی، از آن جایی که تحلیل AMC-rtb از تحلیل SMC دقیق‌تر است، انتظار می‌رود که قدرت زمان‌بندی AMC-rtb بیشتر از SMC باشد. ارزیابی‌ها نیز در شکل ۲ همین را اثبات می‌کنند.

<sup>۱</sup> Adaptive Mixed Criticality – Response Time Bound

### AMC-max<sup>۱</sup> - ۳ - ۲ - ۲ شرط کافی

در این تحلیل تلاش می‌شود که قسمت‌هایی که در تحلیل AMC-rtb فرضیاتی سخت‌گیرانه داشته‌اند، بهبود یابند و حد بالای دقیق‌تری از بدترین زمان پاسخگویی وظایف به دست بیاید. واژه‌ی max در این نام‌گذاری، به بیشترین میزان تداخل-میان وظایف LO و HI- اشاره دارد.

فرض می‌شود که اگر تغییر در حالت سامانه در لحظه‌ی  $s$  توسط وظیفه  $s$  صورت گیرد، تداخلی که وظیفه  $s$  برای وظیفه‌ی  $i$  از درجه‌ی بحرانی بالا ایجاد می‌کند، چقدر است. این مقدار با  $R_i^s$  نمایش داده می‌شود که نحوه محاسبه‌ی آن در رابطه‌ی ۳-۷ آورده شده‌است و برابر است با بیشترین زمان پاسخگویی وظیفه‌ی  $i$  در صورتی که تبدیل حالت در لحظه‌ی  $s$  اتفاق بیافتد. بیشترین مقدار ممکن برای  $R_i^s$ ، حد بالای زمان پاسخگویی وظیفه‌ی  $i$  را به دست می‌دهد که در رابطه‌ی ۳-۸ با  $R_i^*$  نشان داده شده است [۶].

$$R_i^s = C_i(H) + \sum_{\tau_j \in \text{hpL}(i)} \left( \left\lceil \frac{s}{T_j} \right\rceil + 1 \right) C_j(LO) \quad (7-3)$$

$$+ \sum_{\tau_k \in \text{hpH}(i)} \left\{ M(k, s, R_i^s) C_k(HI) + \left( \left\lceil \frac{t}{T_k} \right\rceil - M(k, s, R_i^s) \right) C_k(LO) \right\}$$

$$M(k, s, R_i^s) = \min \left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\} \quad (8-3)$$

$$R_i^* = \max(R_i^s) \quad \forall s$$

بررسی‌های باروآح و همکارانش برای دقیق‌تر کردن رابطه‌ی ۳-۷ از حیطه‌ی این گزارش خارج است اما می‌توان آن را به صورت کلی توضیح داد؛ برای محاسبه‌ی  $R_i^s$  میزان تداخل وظایف LO و میزان تداخل وظایف HI بر روی وظیفه‌ی  $i$  به طور جداگانه محاسبه می‌شود. در محاسبه‌ی تداخل وظایف LO فرض می‌شود که وظیفه‌ی  $s$  که باعث تبدیل حالت شده‌است، قبل از تبدیل حالت سامانه، به طور کامل اجرا می‌شود. در محاسبه‌ی تداخل وظایف HI نیز تلاش شده‌است با ارائه‌ی فرمول  $M(k,s,t)$  تعداد دقیق وظایفی که قبل از لحظه‌ی  $s$  اجرای آن‌ها کامل

<sup>۱</sup> Adaptive Mixed Criticality – Maximum Interference

می‌شود و در نتیجه با  $C(LO)$  اجرا می‌شوند از تعداد وظایفی که بعد از لحظه‌ی  $s$  و با  $C(HI)$  اجرا می‌شوند، تمیز داده شود. مقادیری از  $s$  که می‌توانند  $R_i^S$  را بیشینه کنند، احتمالاً زمان‌های انتشار وظایف ( $hpL(i)$  هستند [۶].

تحلیل AMC-max که رابطه‌ی ۳-۸ نشان داده شد، از تحلیل AMC-rtb بسیار دقیق‌تر است اما همچنان فرضیات سخت‌گیرانه‌ای دارد که باعث می‌شود این تحلیل نیز یک شرط کافی باشد و نتواند آزمون دقیقی برای بررسی زمانبندی‌پذیری مجموعه وظایف بحرانی مختلط با الگوریتم AMC ارائه دهد [۶].

### ۳ - ۳ - زمانبندی اولویت پویا با الگوریتم EDF

می‌دانیم که الگوریتم EDF در زمانبندی سامانه‌های بی‌درنگ سنتی یک الگوریتم بهینه است و آزمون دقیقی بر اساس میزان بهره‌وری وظایف دارد [۵]، به همین دلیل بررسی این الگوریتم برای زمانبندی سامانه‌های بحرانی مختلط نیز انجام شده است. در سامانه‌های بی‌درنگ سنتی، الگوریتم‌های اولویت پویایی چون EDF توانایی بالاتری برای زمانبندی مجموعه وظایف دارند. از این رو انتظار می‌رفت که این الگوریتم در زمانبندی سامانه‌های بحرانی مختلط نیز عملکرد بهتری از الگوریتم‌های اولویت ثابت داشته باشد. وستال و باروآح در تحقیقی نشان دادند که ترکیبی از الگوریتم اولویت ثابت و EDF برای زمانبندی بحرانی مختلط، عملکرد بهتری از SMC-NO که پیشتر توسط وستال ارائه شده بود دارد اما این الگوریتم که آن را زمانبندی اولویت-ترکیبی<sup>۱</sup> نامیده بودند نیز الگوریتم بهینه‌ای نبود و نمونه‌هایی از مجموعه وظایف امکان‌پذیر وجود داشتند که با این الگوریتم قابل زمانبندی نبودند [۱].

### ۳ - ۳ - ۱ - اثبات ناکارآمدی EDF

با فرض اینکه برای زمانبندی‌پذیر بودن یک سامانه بحرانی مختلط، حالت سنتی آن نیز باید امکان‌پذیر باشد، وستال و باروآح با مجموعه وظایف نمونه‌ی جدول ۵ نشان دادند که الگوریتم EDF برای زمانبندی سامانه‌های بحرانی مختلط بهینه نیست [۱].

<sup>۱</sup> Hybrid-Priority

جدول ۵ - نمونه مجموعه وظایف بحرانی مختلط ارائه شده برای اثبات ناکارآمدی الگوریتم EDF [۱]

i	D	T	L	C(LO)	C(HI)
1	4	4	2 (HI)	2	2
2	7	7	1 (LO)	2	5

وظیفه ۱ یک وظیفه با بحرانیت بالاتر است. با اختصاص دادن اولویت بیشتر به وظیفه ۱ نسبت به وظیفه ۲، می‌توان تأیید کرد که هر دو وظیفه در سطوح بحرانیت مورد نظر خود، مهلت‌های خود را برآورده می‌کنند؛ بنابراین، سامانه با یک الگوریتم اولویت ثابت قابل زمان‌بندی است [۱].

اکنون EDF را در نظر بگیرید و بباید روى کار دوم از وظیفه ۱ تمرکز کنیم. به یاد داشته باشید که معنای مدل وظیفه چند بحرانیتی ایجاب می‌کند که تمام مقادیر WCET مورد استفاده برای تضمین اینکه یک وظیفه مهلت خود را برآورده می‌کند، همان سطح اطمینان مورد نیاز توسط وظیفه را داشته باشد. وظایفی که اولویت بیشتری نسبت به کار دوم از وظیفه ۱ دارند شامل کار اول از وظیفه ۲ و به طور غیر مستقیم، کار اول از وظیفه ۱ هستند. در یک سامانه بحرانی مختلط، باید از مقادیر WCET های  $C_1$  و  $C_2$  برای تعیین اینکه آیا وظیفه ۱ مهلت‌های خود را برآورده می‌کند، استفاده شود. اما در شبیه‌سازی EDF با این تخمین‌های WCET، به راحتی می‌توان دید که کار دوم از وظیفه ۱ در لحظه زمانی ۸ مهلت خود را از دست می‌دهد (از آنجا که کار اول از وظیفه ۱ در بازه [۰, ۲) و کار اول از وظیفه ۲ در بازه [۲, ۷) اجرا می‌شوند، در نتیجه فقط یک واحد اجرایی، به جای دو واحد مورد نیاز، برای کار دوم از وظیفه ۱ قبل از مهلتش باقی می‌ماند) [۱].

### ۳ - ۳ - ۲ تحقیقات انجام‌شده برای گسترش EDF

پس از باروآح و وستال، بررسی‌های بیشتری روی الگوریتم EDF انجام شد. به عنوان مثال، پارک و کیم<sup>۱</sup> نسخه تغییر یافته‌ای از آن را تحت عنوان CBEDF<sup>۲</sup> ابداع کردند. سپس، اکبرگ و بی<sup>۳</sup> مدلی تغییر یافته از الگوریتم EDF را ارائه دادند که شباهت‌های زیادی به EDF-VD داشت. این الگوریتم که بر پایه اولویت‌بندی ثابت استوار است، برای هر وظیفه HI، دو اولویت واقعی و مجازی را در نظر می‌گیرد. در حالت عادی فعالیت سامانه، وظایف HI با

<sup>۱</sup> Park and Kim

<sup>۲</sup> Criticality Based Earliest Deadline First

<sup>۳</sup> Ekberg and Yi

اولویت مجازی زمانبندی می‌شوند و در صورت تغییر حالت سامانه، وظایف LO حذف شده و وظایف HI با اولویت واقعی زمانبندی می‌شوند. این الگوریتم برای بیش از دو سطح بحرانیت و همچنین برای مدل‌های وظیفه‌ای که در آن‌ها عواملی به جز WCET تغییرپذیر بودند، تعمیم داده شد. پس از بررسی‌های بیشتر روی الگوریتم‌های ابداعی و همچنین ویرایش آزمون زمانبندی EDF و ارائه الگوریتم‌هایی مانند QPA که همگی بر اساس ایجاد مهلت‌های مجازی برای وظایف بودند، بارواح و همکارانش در سال ۲۰۱۱، الگوریتم EDF-VD را برای زمانبندی سامانه‌های بحرانی مختلط دو بحرانیتی با مهلت ضمنی ارائه دادند. پس از آن، تحقیقات بر روی EDF-VD ادامه یافت و برای مدل‌های وظیفه‌ای با مهلت و تناوب متغیر و همچنین  $k$  سطح بحرانیت تعمیم یافت [۳].

لیپاری و بوتازو<sup>۲</sup> با استفاده از روش‌های مبتنی بر رزرواسیون<sup>۳</sup> و تغییر الگوریتم EDF، الگوریتمی را ارائه می‌دهند که مهلت وظایف HI را به گونه‌ای تغییر می‌دهد که ظرفیت باقی‌مانده برای اجرای وظایف LO را بیشینه کند. سو<sup>۴</sup> و همکارانش نیز روش‌هایی مبتنی بر مدل وظیفه‌ی ارجاعی را معرفی کرده است. در این روش‌ها، برای افزایش قابلیت زمانبندی‌پذیری مجموعه وظایف، ممکن است در یک یا چند ویژگی آن‌ها تغییراتی اعمال شود. سو در روش خود، با افزایش تناوب وظایف LO (کاهش فرکانس آن‌ها)، سطح سرویس حداقلی را برای وظایف LO در نظر می‌گیرد. در صورتی که پس از اجرای وظایف HI، زمان اضافی باقی بماند، فرکانس وظایف LO افزایش می‌یابد. تحلیل جایگزین برای سامانه‌های بحرانی مختلط زمانبندی شده توسط EDF توسط مهدیانی و مسورو و همچنین سانتینلی<sup>۵</sup> و همکاران ارائه شده است. آن‌ها از چندین منحنی محدودیت تقاضا<sup>۶</sup> استفاده می‌کنند تا تحلیل حساسیتی را ارائه دهند که می‌تواند برای ایجاد تعادل بین استفاده از منابع و قابلیت زمانبندی اعمال شود. اشمیت<sup>۷</sup> و همکاران نیز از مقیاس‌بندی غیر یکنواخت مهلت‌ها<sup>۸</sup> برای بهبود کیفیت خدمات سامانه‌های زمانبندی شده توسط EDF استفاده می‌کنند [۳].

<sup>۱</sup> Quick convergence Processor-Demand Analysis

<sup>۲</sup> Lipari and Buttazzo

<sup>۳</sup> Reservation-Based

<sup>۴</sup> Su

<sup>۵</sup> Santinelli

<sup>۶</sup> Demand-Bound Curve

<sup>۷</sup> Schmidt

<sup>۸</sup> Non-Uniform Deadline Scaling

### ۳ - ۳ - ۳ تبدیل EDF به EDF-VD

از بین تمامی الگوریتم‌های ارائه شده بر اساس EDF که در بالا به آن‌ها اشاره شد، الگوریتم EDF-VD شناخته‌شده‌تر است که به دلیل کیفیت بهتر آن نسبت به سایرین، توجه بیشتری را دریافت کرده است [۳]. این الگوریتم ابتدا برای مجموعه‌ای از کارهای مستقل ارائه شد [۷] و پس از آن برای مجموعه‌ای از وظایف بی‌درنگ اسپورادیک که تعداد نامحدودی کار را منتشر می‌کنند، تعمیم داده شد [۴]. در ادامه به بررسی این الگوریتم بر روی مجموعه‌ای از وظایف اسپورادیک می‌پردازیم.

**۳ - ۳ - ۱ گسترش تعاریف مفاهیم زمان‌بندی بی‌درنگ برای بحرانی مختلط**  
ابتدا تعاریف زمان‌بندی را برای یک مجموعه وظایف بحرانی مختلط طبق تعریف ۲-۳ و ۳-۳ گسترش می‌دهیم.  
این تعاریف، گسترشی از تعریف ۴-۲ و ۵-۲ که در بخش ۲ - ۱ - ۲ ارائه شدند، هستند.

**تعریف ۲-۳: زمان‌بندی پذیری در سامانه‌های بحرانی مختلط**، یا به اختصار MC-Schedulability، آن است که یک الگوریتم (برخط) یک سامانه وظیفه‌ای اسپورادیک  $\tau$  را به طور صحیح زمان‌بندی می‌کند اگر بتواند هر دنباله‌ی کار تولید شده توسط  $\tau$  را به گونه‌ای زمان‌بندی کند که اگر سطح بحرانیت سناریوی مربوطه  $\Gamma$  باشد، آنگاه تمام وظایف با سطح حداقل  $\Gamma$  بین زمان انتشار و مهلت خود تکمیل شوند [۴].

**تعریف ۳-۳: یک سامانه (مجموعه وظایف) زمان‌بندی پذیر در بحرانیت مختلط**، یا به اختصار MC-Schedulable نامیده می‌شود اگر الگوریتم صحیحی برای زمان‌بندی آن وجود داشته باشد [۴].

از آنجایی که الگوریتم EDF دارای تحلیلی مبتنی بر بهره‌وری سامانه می‌باشد، لازم است تا تعاریف بهره‌وری را نیز برای سامانه‌های بحرانی مختلط گسترش دهیم.

**تعریف ۴-۴: بهره‌وری وظیفه**  $\tau$  در سامانه‌ای با سطح بحرانیت عملیاتی  $\Gamma$  و سرعت پردازشی  $\sigma$  در واحد زمان، از طریق رابطه‌ی ۹-۳ محاسبه می‌شود [۴]:

$$u_j(\Gamma) = \frac{c_j(k)}{T_j \times \sigma} \quad (j = 1, \dots, n) \quad (9-3)$$

**تعریف ۵-۵: بهره‌وری مجموعه‌ی وظایف با درجه‌ی بحرانیت  $\alpha$**  در سامانه‌ای با سطح بحرانیت عملیاتی  $\Gamma$  و سرعت پردازشی  $\sigma$  در واحد زمان، از طریق رابطه‌ی ۱۰-۳ محاسبه می‌شود [۴]:

$$U_i(\Gamma) = \sum_{j \in L_i} u_j(\Gamma), \quad (k = 1, \dots, i) \quad (10-3)$$

می‌دانیم که در مورد یک سطح بحرانیت واحد (سامانه‌های سنتی)، یک مجموعه وظایف با مهلت ضمنی بر روی  $m$  پردازنه با سرعت  $\sigma$  قابل اجرا است اگر و تنها اگر  $m \leq U_1(1)$  و برای همهی  $j$ ها  $1 \leq u_j(1) \leq U_1(1)$  باشد. با تعمیم این مفهوم به سامانه‌های بحرانی مختلط، می‌توانیم به شرط لازم برای MC-Schedulability برسیم. برای سامانه‌های بحرانی مختلط مشکل از  $K$  درجه‌ی بحرانیت، سامانه زمان‌بندی‌پذیر است اگر رابطه‌ی  $11-3$  برقرار باشد [۴]:

$$\sum_{i=k}^K U_i(k) \leq 1 \quad (11-3)$$

### ٣ - ٣ - ٣ - ٢ نحوه عملکرد الگوریتم EDF-VD

اگر رابطه‌ی  $11-3$  برقرار باشد به سراغ زمان‌بندی سامانه با الگوریتم EDF می‌رویم. الگوریتم EDF-VD که متغیری از الگوریتم EDF است برای سامانه‌ای دو-بحرانیتی که روی پردازنه‌ی تک‌هسته‌ای با سرعت واحد اجرا می‌شود، به صورت زیر و در دو حالت عمل می‌کند [۴]:

(۱) اگر  $1 \leq U_1(1) + U_2(2)$ ، آنگاه الگوریتم EDF را برای مهلت‌های تغییر نیافته (اصلی) وظایف اعمال کنید. به محض رسیدن سامانه به سطح ۲، یعنی اجرای یک وظیفه بیش از WCET خود در سطح ۱، وظایف موجود در سطح ۱ را لغو کنید.

(۲) اگر حالت ۱ برقرار نباشد و رابطه‌ی  $1 \leq U_1(1) + \frac{U_2(1)}{1-U_2(2)}$  برقرار باشد، آنگاه  $\lambda = \frac{U_2(1)}{1-U_2(2)}$  را تنظیم کنید. سپس در حالی که سامانه در سطح ۱ است، برای وظایف سطح ۲، مقدار  $\lambda \times T_j \times \hat{T}_j$  را تعریف کنید. مهلت‌های کارهای منتشرشده توسط وظایف سطح ۲ را با افزودن  $\hat{T}_j$  به زمان انتشار هر کدام بازتعریف کنید (مهلت مجازی). مهلت‌های کارهای وظایف موجود در سطح ۱ را همانطور که بوده‌اند، باقی بگذارید و EDF را برای مهلت‌های جدید اعمال کنید.

به محض رسیدن سامانه به سطح ۲، وظایف موجود در سطح ۱ را لغو کنید؛ مهلت‌های هر کار از وظایف سطح ۲ را با افزودن  $T_j$  به زمان انتشار آن‌ها بازنگشانی کنید؛ EDF را برای این مهلت‌های (واقعی) اعمال کنید.

بنابراین شرط کافی برای آنکه یک سامانه دو-بحرانیتی با استفاده از الگوریتم EDF-VD روی پردازنده‌ای تک‌هسته‌ای زمان‌بندی‌پذیر یا MC-Schedulable باشد، به شکل رابطه‌ی ۱۲-۳ تعریف می‌شود [۴].

$$U_1(1) + \min\left(U_2(2), \frac{U_2(1)}{1 - U_2(2)}\right) \leq 1 \quad (12-3)$$

باروآح و همکارانش این رابطه را برای  $k$  سطح بحرانیت نیز تعمیم داده‌اند و آن را با نام EDF-VD(K) معرفی می‌کنند اما بررسی آن الگوریتم از شرح این گزارش خارج است [۴].

### ۳ - ۳ - ۳ - اثبات شرط کافی الگوریتم EDF-VD

در اینجا رابطه‌ی ۱۲-۳ را به عنوان یک شرط کافی برای تحلیل زمان‌بندی‌پذیری یک مجموعه وظایف دو-بحرانیتی بر روی پردازنده‌ای تک‌هسته‌ای اثبات می‌کنیم. برای اثبات این رابطه باید آن را برای هر دو حالت الگوریتم EDF-VD اثبات کنیم اما حالت اول که  $1 \leq U_2(2) + U_1(1)$ ، یک حالت بدیهی است، پس به سراغ اثبات حالت دوم که  $1 \leq U_1(1) + \frac{U_2(1)}{1 - U_2(2)}$  در سناریوهای مختلف می‌رویم [۴].

#### سناریوی اول: سامانه در سطح ۱ (LO) قرار دارد

در این سناریو باید نشان دهیم که سامانه با وجود اینکه مهلت وظایف سطح ۲ تغییر کرده‌است، همچنان در سطح ۱ قابل زمان‌بندی می‌باشد. به این منظور ثابت می‌کنیم:

$$\begin{aligned} \sum_{i \in L_1} \frac{c_i(1)}{T_i} + \sum_{i \in L_2} \frac{c_i(1)}{\lambda T_i} &= \sum_{i \in L_1} \frac{c_i(1)}{T_i} + \sum_{i \in L_2} \frac{c_i(1)}{\frac{U_2(1)}{1 - U_1(1)} T_i} = U_1(1) + \frac{1 - U_1(1)}{U_2(1)} \sum_{i \in L_2} \frac{c_i(1)}{T_i} \\ &= U_1(1) + \frac{1 - U_1(1)}{U_2(1)} U_2(1) = 1 \end{aligned}$$

#### سناریوی دوم: سامانه در سطح ۲ (HI) قرار دارد

در این حالت، سامانه تنها شامل وظایف سطح ۲ می‌باشد که با مهلت اصلی خودشان زمان‌بندی شده‌اند پس باید نشان دهیم که  $1 \leq U_2(2)$ . اثبات این مورد نیز از روی رابطه‌ی ۱۱-۳ و ۱۲-۳ بدیهی است و می‌توانیم نتیجه بگیریم که  $1 \leq U_2(2)$ .

#### سناریوی سوم: سامانه از حالت ۱ (LO) به حالت ۲ (HI) می‌رسد

برای اثبات این سناریو باید نشان دهیم که مجموعه وظایفی که با مهلت‌های مجازی زمان‌بندی کرده‌ایم، با رسیدن سامانه به حالت ۲ در هر لحظه‌ی دلخواه  $t^*$ ، زمان کافی برای اجرای کامل کارهای فعال (کارهایی که قبل از رسیدن به سطح ۲ منتشر شده‌اند و هنوز کامل نشده‌اند) از وظایف سطح ۲ قبل از فرارسیدن مهلتشان را دارد.

یک کار فعال دلخواه را با زنایش می‌دهیم. مهلت اصلی این کار برابر است با  $d_j = r_j + T_j$  اما این کار با استفاده از مهلت مجازی  $\hat{d}_j = r_j + \hat{T}_j$  زمان‌بندی شده‌است. از آنجایی که این کار در لحظه‌ی  $t^*$  فعال است می‌توان نتیجه گرفت که  $t^* \leq \hat{d}_j$ .

$$d_j - \hat{d}_j = (r_j + T_i) - (r_j + \hat{T}_i) = T_i - \lambda T_i \quad (13-3)$$

رابطه‌ی ۱۳-۳ بدان معناست که در زمان  $t^*$  برای هر  $j$  حداقل  $(1 - \lambda)T_i$  واحد زمان باقی مانده است تا کارش را به موقع تمام کند. حالا مساله به یک مساله تک‌بحرانیتی در سامانه‌ای با تناوب‌هایی که اندازه‌ی هر کدام  $(1 - \lambda)T_i$  است و فقط از کارهایی با درجه‌ی بحرانی ۲ تشکیل شده، تبدیل می‌شود و باید نشان دهیم که بهره‌وری این سامانه از ۱ کمتر است تا با EDF قابل زمان‌بندی باشد. که بدین معناست که سامانه اصلی می‌تواند از زمان  $t^*$  به بعد به درستی زمان‌بندی شود. از حالت دوم از رابطه‌ی ۱۲-۳ می‌توانیم ثابت کنیم:

$$\frac{U_2(1)}{1 - U_2(2)} \leq 1 - U_1(1) \Rightarrow \lambda = \frac{U_2(1)}{1 - U_1(1)} \leq 1 - U_2(2) \Rightarrow 1 - \lambda \geq U_2(2)$$

پس بهره‌وری سامانه‌ی جدید از ۱ کمتر است:

$$\sum_{i \in L_2} \frac{c_i}{(1 - \lambda)T_i} \leq \sum_{i \in L_2} \frac{c_i}{U_2(2)T_i} = 1$$

### ۳ - ۳ - ۴ - مقایسه‌ی الگوریتم EDF-VD با سایر الگوریتم‌ها

همانطور که در بخش ۳ - ۱ اشاره شد، نشان داده شده است که مسئله قابلیت زمان‌بندی بحرانیت مختلط چه به صورت قبضه‌ای چه غیر از آن، حتی اگر فقط دو سطح بحرانیت وجود داشته باشد، به شدت NP-Hard است.

بنابراین فقط تحلیل کافی و نه دقیق امکان‌پذیر است. گرچه در سال ۲۰۱۷ کاهیل<sup>۱</sup> و همکاران ادعا کردند که مثال نقضی برای اثبات اینکه مسئله بهینگی بحرانیت مختلط به کلاس NP تعلق دارد یافته‌اند اما هنوز هیچ الگوریتم بهینه‌ای برای زمان‌بندی سامانه‌های بحرانی مختلط یافت نشده‌است [۳].

برای رویکردها و آزمایش‌هایی که فقط کافی هستند، ارزیابی کیفیت آن‌ها با ضریبی تحت عنوان ضریب افزایش سرعت امکان‌پذیر است.

### ۳ - ۴ - ۱ ضریب افزایش سرعت<sup>۲</sup>

یک ضریب سرعت به میزان  $X$  به طوری که  $X$  بزرگ‌تر از ۱ باشد، برای آزمون قابلیت زمان‌بندی الگوریتم  $S$  به این معناست که مجموعه وظایفی که بر روی یک پردازنده با سرعت ۱ قابل زمان‌بندی نیست، اگر سرعت پردازنده به  $X$  افزایش یابد، توسط  $S$  قابل زمان‌بندی تلقی خواهد شد [۳].

البته، به طور کلی، امکان ندارد بدانیم که آیا مجموعه وظایف بر روی پردازنده اصلی با سرعت ۱ قابل زمان‌بندی است چون این نیازمند یک آزمون دقیق است، اما یک طرح و آزمون زمان‌بندی واقعی با ضریب سرعت مثلاً ۲ قطعاً بهتر از یکی با ضریب سرعت ۱۰ است [۳].

می‌توان این مفهوم را با یک مثال به شکل دقیق‌تری توضیح داد:

- زمان‌بندی یک مجموعه از کارها با الگوریتم PC دارای ضریب افزایش سرعت نامحدود است. این نتیجه به راحتی با در نظر گرفتن یک سامانه که دارای دو کار است نشان داده می‌شود. کار با بحرانیت LO زمان محاسبه کوچکی به اندازه ۱، و مهلت ۲ دارد. کار با بحرانیت HI زمان محاسبه عظیمی به اندازه  $G$  و مهلت  $1 + G$  دارد. این دو کار هر دو مهلت‌های خود را برآورده می‌کنند اگر به وظیفه با بحرانیت LO بالاترین اولویت داده شود. اما طبق الگوریتم‌های PC، کار HI حتما باید اولویت بالاتری از کار LO داشته باشد. در این صورت سامانه تنها در حالتی قابل زمان‌بندی خواهد بود که پردازنده بتواند  $1 + G$  کار را قبل از مهلت ۲ انجام دهد. نتیجتا، ضریب افزایش سرعت باید  $\frac{G+1}{2}$  باشد. از آنجا که  $G$  می‌تواند به اندازه دلخواه بزرگ باشد، ضریب افزایش سرعت نیز به طور مؤثر نامحدود است [۳].

<sup>۱</sup> Kahil

<sup>۲</sup> Speedup Factor

### ۳ - ۳ - ۲ - ۴ ضریب افزایش سرعت EDF-VD

برای سامانه‌های زمانبندی شده توسط EDF، باروآح و همکاران ثابت می‌کنند که EDF-VD نیز بر روی یک پردازنده تک‌هسته‌ای که سرعت آن با ضریب  $\phi = 1.618$  (نسبت طلایی) افزایش یافته است، قابل زمانبندی است. آن‌ها همچنین نشان می‌دهند که یک مجموعه متناهی از کارهای مستقل که بر روی  $m$  پردازنده یکسان زمانبندی شده است، با ضریب سرعت  $\frac{1}{m} + \phi$  قابل زمانبندی است [۴]. سپس نشان می‌دهند که در یک سامانه بخش‌بندی شده، ضریب سرعت  $\epsilon + \phi$  برای هر مقدار  $\epsilon > 0$  قابل دست‌یابی است. در کارهای بعدی، این حد را از  $\phi$  به  $\frac{3}{4}$  بهبود می‌بخشند. تحقیقات در زمینه‌ی ضریب افزایش سرعت EDF-VD توسط دیگر محققان نیز ادامه داده شده‌است [۳].

### ۳ - ۳ - ۳ - ۴ ضریب افزایش سرعت الگوریتم‌های رقیب

برای زمانبندی کارهای مستقل با الگوریتم اولویت ثابت، یک الگوریتم اختصاص اولویت و تحلیل آن با ضریب افزایش سرعت  $SL$  (برای  $L$  سطح بحرانیت) یافت شده است، که در آن  $SL$  ریشه معادله  $(1+x)^{L-1} = x^L$  است. برای سامانه‌های دو-بحرانیتی، نتیجه برابر با نسبت طلایی  $= 1.618$  است که این مقدار با ضریب افزایش سرعت EDF-VD برابر است [۳]. نکته‌ی حائز اهمیت در این مقایسه، آن است که EDF-VD علاوه بر زمانبندی کارهای مستقل، قادر به زمانبندی یک مجموعه وظایف اسپورادیک نیز هست.

رویکرد MC-Fluid نیز دارای ضریب سرعت  $\frac{3}{4}$  است اما هنوز هیچ الگوریتم دیگری ارائه نشده است که ضریب افزایش سرعتی بهتر از  $\frac{3}{4}$  داشته باشد [۳].

### ۳ - ۳ - ۵ محدودیت‌های EDF-VD

با وجود آنکه الگوریتم EDF-VD الگوریتم قدرتمندی است که با یک ضریب افزایش سرعت قابل توجه قادر به زمانبندی سامانه‌های بحرانی مختلط است، اما معايب و محدودیت‌هایی دارد که لازم است به شمار آورده شوند:

- در این الگوریتم فرض بر آن است که در صورت وقوع خطای زمانی، همه‌ی وظایف با درجه بحرانیت پایین، کنار گذاشته می‌شوند در صورتی که همانطور که در بخش ۲ - ۲ - ۳ به آن اشاره شد، چنین عملکردی برای صنعتی‌سازی زمانبندی سامانه‌های بحرانی مختلط مورد پذیرش نیست چرا که وظایف با درجه بحرانیت پایین نیز مهلت و هدف خاصی دارند و باید به گونه‌ای اجرا شوند.
- اگر کاری از هر کدام از وظایف از بدترین زمان اجرای خوش‌بینانه خود (C(LO)) تجاوز کند، سامانه به حالت بحرانی بالا منتقل می‌شود و فرض می‌شود که تمامی وظایف با بحرانیت بالا رفتاری بدینانه از خود

بروز خواهند داد. این فرض نیز بیش از حد بدینانه است؛ زیرا تغییرات حالت تمامی وظایف، مستقل از یکدیگر است.

- همچنین در حین تغییر حالت سامانه و رسیدن به درجه‌ی بحرانی بالا، فرض می‌شود که تمامی وظایف با بحرانیت بالا در زمان اجرای بدینانه خود ((HI))، اجرا خواهند شد. این فرض نیز بسیار بدینانه است چرا که در واقعیت به ندرت پیش می‌آید که یک وظیفه به این حد از زمان اجرا برسد. این امر موجب تخصیص بیش از حد منابع به وظایف با درجه‌ی بحرانی بالا و هدر رفت منابع می‌شود، دقیقاً بر عکس هدفی که سامانه‌های بحرانی مختلط تلاش دارند به آن دست یابند.

تمامی حالات فوق بسیار بدینانه هستند و از معایب الگوریتم EDF-VD به حساب می‌آیند. علت بروز این معایب آن است که این الگوریتم سازوکارهای مشخصی برای کاهش نظاممند سطح سرویس وظایف از درجه‌ی بحرانیت پایین ندارد و آن‌ها را به یکباره از سامانه حذف می‌کند. همچنین سازوکار نظاممندی برای افزایش تدریجی منابع اختصاص داده شده به وظایف با درجه‌ی بحرانی بالا وجود ندارد. گرچه تمام این سازوکارها می‌توانند در زمان اجرا اعمال شوند، اما به کارگیری آن‌ها در تحلیل‌های زمان‌بندی EDF-VD تاثیرگذار خواهد بود و نیاز به محاسباتی بسیار دقیق‌تر و پیچیده‌تر دارد.

## ۴ - پیاده‌سازی چارچوب زمان‌بندی

این نرم‌افزار از دو بخش تشکیل شده است؛ بخش اول آن یک مجموعه‌ی وظایف بحرانی مختلط را با مقادیر تصادفی تولید می‌کند و بخش دوم و اصلی آن یک چارچوب برای زمان‌بندی و شبیه‌سازی اجرای وظایف بحرانی مختلط است که با دریافت فایل‌های ورودی وظایف و تنظیمات شبیه‌سازی، اجرای وظایف بحرانی مختلط را بر روی یک پردازنده تک‌هسته‌ای با سرعت‌های مختلف شبیه‌سازی می‌کند و زمان‌بندی دقیق وظایف را بر اساس الگوریتم EDF-VD به همراه داده‌های آماری زمان‌بندی، خروجی می‌دهد. تمامی الگوریتم‌های این سامانه مطابق توصیفات موجود در فصل ۳ پیاده‌سازی شده‌اند. در این بخش، با رویکردی بالا به پایین یا کل به جزء، روش توسعه این نرم‌افزار توضیح داده خواهد شد. دقت داشته باشید که در سراسر این سامانه منظور از عدد صحیح، اعداد صحیح بزرگتر از صفر و منظور از عدد گویا، اعداد گویای بزرگتر از صفر با دقت ۰.۱ می‌باشد.

## ۴ - ۱ مدل سامانه بحرانی مختلط

در این پیاده‌سازی یک سامانه بحرانی مختلط دو-بحرانیتی با وظایف اسپورادیک قبضه‌ای دارای مهلت‌های ضمنی روی یک پردازنده‌ی تک‌هسته‌ای با سرعت و فرکانس متغیر، مدل شده است. در این پیاده‌سازی فرض می‌شود که وظایف هیچ وابستگی‌ای به هم ندارند و تقدم و تاخر اجرای هر کدام بر دیگری تاثیری ندارد؛ همچنین تنها منبع مشترک بین وظایف زمان و توان پردازشی می‌باشد. در این سامانه هر وظیفه به صورت  $\tau_i = (T_i, \vec{C}_i, L_i, \varphi_i)$  تعریف می‌شود که به ترتیب  $T_i$  تناوب و به صورت ضمنی مهلت نسبی یا  $D_i$  وظیفه را مشخص می‌کند و می‌تواند با اعداد طبیعی مقداردهی شود،  $\vec{C}_i$  بدترین زمان‌های اجرای بدینانه و خوش‌بینانه یک وظیفه را به صورت  $(HI)_i$  برای حالت بدینانه و  $(LO)_i$  برای حالت خوش‌بینانه مشخص می‌کند و می‌تواند با اعداد گویای بزرگتر از صفر و با دقت ۰.۱ مقداردهی شود،  $L_i$  درجه‌ی بحرانیت یک وظیفه را مشخص می‌کند و می‌تواند با مقادیر ۰ یا HI تعريف شود و در نهایت  $\varphi_i$  فاز یک وظیفه را مشخص می‌کند که می‌تواند با اعداد طبیعی یا صفر مقداردهی شود. البته در مدل‌سازی مجموعه وظایف اسپورادیک، عموماً فاز کاربردی ندارد اما برای آنکه این پیاده‌سازی بتواند وظایف متناوب را نیز در حالت‌های متفاوت پشتیبانی کند، امکان تعريف فاز برای هر تسك وجود دارد و گرنه این مقدار به صورت پیشفرض صفر در نظر گرفته می‌شود. همچنین شناسه‌ی یکتای یک وظیفه از روی ۱ ساخته می‌شود.

هر وظیفه می‌تواند در هر لحظه‌ی  $i$   $t = j * T_i + \varphi_i$  (به ازای تمام مقادیر صحیح  $0 \leq j$ ) تعداد نامتناهی کار منتشر کند که هر کدام از این کارها به صورت  $(r_{ij}, \vec{d}_{ij}, C_j, \vec{C}_i, L_i) = (r_{ij}, \vec{d}_{ij}, C_{ij}, \vec{C}_i, L_i)$  تعریف می‌شود. به ترتیب  $r_{ij}$  زمان انتشار یک کار،  $\vec{d}_{ij}$  مهلت‌های واقعی و مجازی یک کار،  $C_j$  مهلت واقعی یک کار،  $\vec{C}_{ij}$  بدترین زمان‌های اجرای یک کار و  $L_i$  درجه‌ی بحرانیت یک کار را مشخص می‌کنند. مقادیر بدترین زمان‌های اجرا و درجه‌ی بحرانیت یک کار با وظیفه‌ی سازنده‌ی آن هم‌مقدار است. شناسه‌ی یکتای یک کار با ترکیبی از  $i$  و  $j$  ساخته می‌شود. مهلت واقعی

یک کار در زمان ساخت آن کار به صورت تصادفی انتخاب می‌شود و بسته به تنظیماتی شبیه‌سازی، مقداری بین ۰ و بدترین زمان اجرای بدینانه آن کار دارد.

## ۴ - ۲ جریان اجرای نرم‌افزارهای تولید وظایف و شبیه‌سازی زمان‌بندی

پیش از هر چیز، توضیح کلایی از جریان اجرای این نرم‌افزار - که به دو بخش کلی تولید فایل‌های ورودی وظایف و شبیه‌سازی زمان‌بندی سامانه‌های بحرانی مختلط تقسیم می‌شود - ارائه می‌دهیم تا خواننده با آن آشنا گردد. سپس با پرداختن به واحدهای تشکیل‌دهنده‌ی هر بخش، توضیح کاملی از این نرم‌افزار ارائه خواهیم داد. به طور کلی می‌توان اجرای این نرم‌افزار را به ۵ مرحله تقسیم کرد:

(۱) ساخت فایل‌های ورودی: این نرم‌افزار علاوه بر دریافت فایل‌های ورودی دلخواه کاربر، قادر است فایل‌های ورودی وظایفش را به طور خودکار تولید کند. به این منظور مقادیری را برای تعداد وظایف، نسبت تعداد وظایف با بحرانیت بالا به تعداد وظایف با بحرانیت پایین، نسبت اندازه‌ی بدترین زمان اجرای بدینانه به اندازه‌ی بدترین زمان اجرای خوش‌بینانه و بهره‌وری مجموعه‌ی وظایف، به عنوان ورودی دریافت می‌کند و با استفاده از الگوریتم‌های *uunifast* و *log\_uniform* یک مجموعه‌ی وظایف بحرانی مختلط با دو سطح بحرانیت و منطبق بر مدل ۴ - ۱ تولید می‌کند.

(۲) دریافت تنظیمات و شروع شبیه‌سازی: بدنه‌ی اصلی این نرم‌افزار که کار شبیه‌سازی مدل وظیفه‌ی ورودی را انجام می‌دهد، می‌تواند با مقادیری گستردگی و به شکلی انعطاف‌پذیر تنظیم شود. این شبیه‌ساز می‌تواند بسته به نیاز کاربر، در حالت سنتی یا حالت بحرانی مختلط اجرا شود. همچنین می‌توان زمان دقیق وقوع تغییر حالت سامانه را برای آن مشخص کرد یا اجازه داد که زمان تغییر حالت سامانه در طول شبیه‌سازی و به صورت خودکار مشخص شود. فرکانس پردازنده، مقدار واحد کار انجام شده در هر تیک از پردازنده و سطحی که سامانه در آن شروع به کار می‌کند نیز قابل تنظیم است. این گستره از تنظیمات ورودی به کاربر اجازه می‌دهد تا یک مجموعه‌ی ثابت از وظایف ورودی را به حالاتی متفاوت زمان‌بندی کند و با توجه به خروجی‌های تولید شده، قادر به مقایسه‌ی تاثیر تنظیمات مختلف بر زمان‌بندی مجموعه‌ی وظایفش باشد. مقادیر دیگری از جمله مدت‌زمان اجرای شبیه‌سازی و همینطور احتمال تجاوز وظایف از {بدترین} زمان اجرای {خوش‌بینانه} آن‌ها<sup>۱</sup> نیز قابل تنظیم هستند. شبیه‌سازی با ساخت وظایف از روی فایل ورودی وظایف، ساخت پردازنده با توجه به فرکانس، ساخت زمان‌بند و همینطور صفت کارهای آماده، آغاز می‌شود

<sup>۱</sup> Overrun Probability

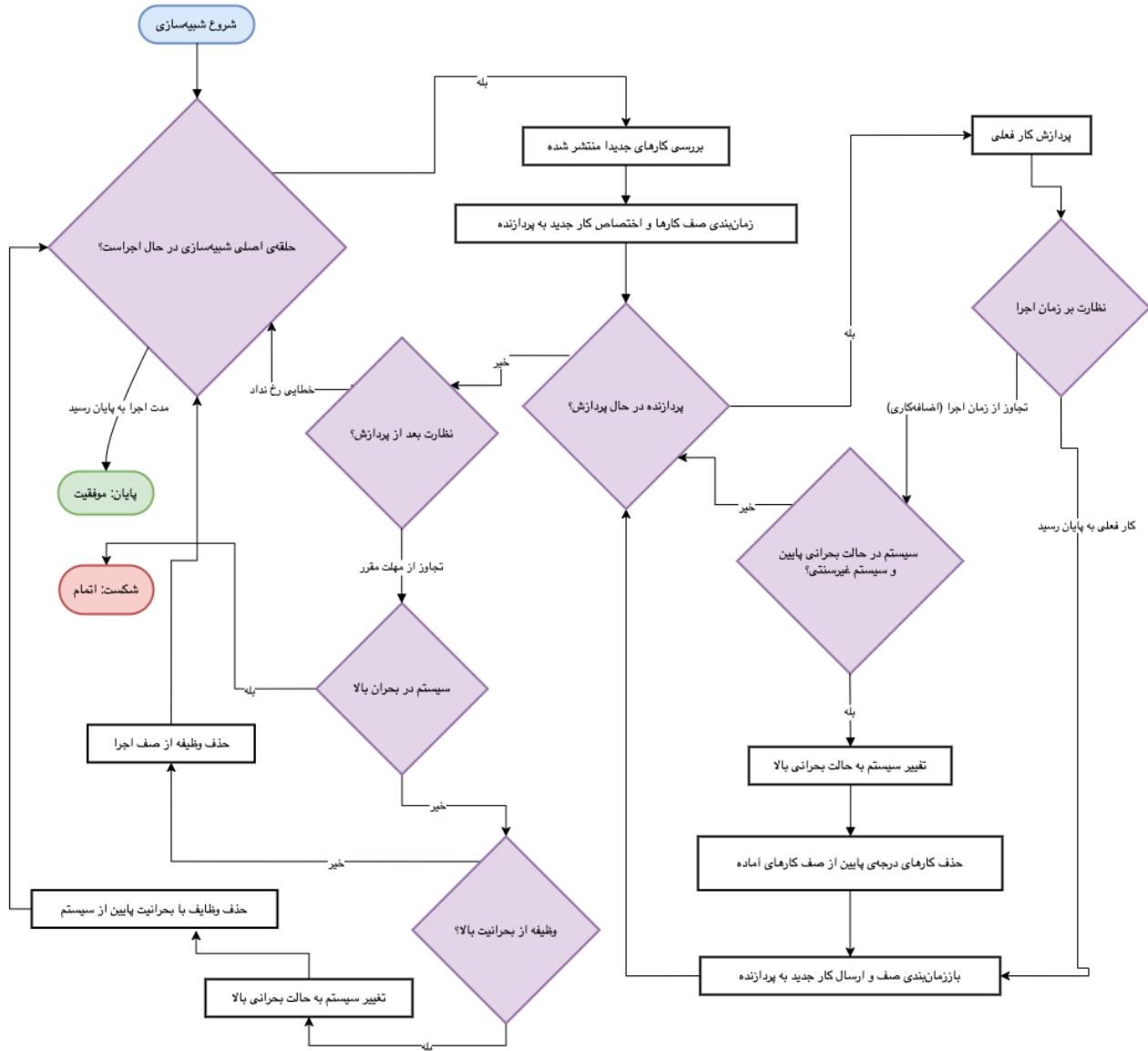
و قبل از آنکه وارد حلقه‌ی اصلی اجرا شود، آنالیزهای لازم و آزمون‌های زمان‌بندی EDF-VD را بر روی مجموعه وظایف اجرا می‌کند.

**۳) اجرای حلقه‌ی اصلی شبیه‌سازی:** این شبیه‌سازی به صورت هم‌گام و در دو حلقه‌ی تو در تو اجرا می‌شود. حلقه‌ی بیرونی از ۰ تا مدت زمان اجرا، تکرار می‌شود و واحد زمان را در این سامانه شبیه‌سازی می‌کند. حلقه‌ی درونی داخل پردازنده قرار دارد و در هر واحد از زمان به تعداد دفعات فرکانس پردازنده تکرار می‌شود. در هر اجرا از حلقه‌ی بیرونی به ترتیب کارها منتشر می‌شوند، در صورتی که کار جدیدی منتشر شده باشد به صفت آماده اضافه می‌شود و صفت آماده مرتب می‌شود، زمان‌بند کاری را برای اجرا به پردازنده اختصاص می‌دهد و پردازنده کاری را که به آن محول شده در یک حلقه‌ی درونی اجرا می‌کند. در این حلقه‌ی درونی نیز سازوکارهایی برای تشخیص اتمام اجرای کارها یا تجاوز کارها از بدترین زمان اجرای آن‌ها در هر تیک وجود دارد. پس از پایان پردازش پردازنده، یک سازوکار نظارت در زمان اجرا بررسی می‌کند که تجاوز از زمان اجرا یا تجاوز از مهلت زمان اجرا رخداده است یا خیر و با اتمام حلقه‌ی بیرونی، زمان سامانه یک واحد به جلو می‌رود.

**۴) مدیریت وقایع حین اجرا:** در صورتی که یک کار تمام شود یا از بدترین مهلت اجرای خود تجاوز کند یا قبل از مهلتش به اتمام نرسد، وقایعی رخ داده‌اند که باید مدیریت شوند. برای تشخیص هر کدام از این وقایع نظارت‌هایی در حین اجرای حلقه‌ی بیرونی شبیه‌سازی و همینطور حلقه‌ی درونی پردازنده تعییه شده‌است. در صورتی که یک کار تمام شود، زمان‌بند باید کار جدیدی را برای پردازنده انتخاب کند. در صورتی که یک کار از مهلت اجرای خود تجاوز کند، سامانه باید در صورت امکان تغییر سطح بددهد و وارد سطحی با درجه بحرانیت بالاتر شود. در صورتی که کاری در مهلت مقرر شد به پایان نرسد، اگر آن کار از درجه‌ی بحرانیت پایین باشد، از صفت کارهای آماده حذف خواهد شد و در صورتی که از درجه‌ی بحرانیت بالا باشد و سامانه نیز در درجه‌ی بحرانیت بالا در حال اجرا باشد، شبیه‌سازی شکست خواهد خورد در غیر این صورت سامانه تغییر حالت خواهد داد و به درجه‌ی بحرانی بالا می‌رود.

**۵) اتمام شبیه‌سازی و تولید خروجی‌ها:** در هر مرحله از اجرای نرمافزار سازوکارهایی تعییه شده‌است که وقوع هر رخدادی را ثبت می‌کند. زمانی که شبیه‌سازی به دلیل تجاوز وظایف بحرانی از مهلت واقعی آن‌ها شکست بخورد یا آنکه مدت زمان اجرای شبیه‌سازی با موقوفیت به پایان نرسد، فایل‌های خروجی سامانه تولید می‌شوند. این فایل‌های خروجی شامل زمان‌بندی سامانه- اختصاص هر کار به زمانی مشخص- و همینطور داده‌های آماری از سامانه هستند؛ به طور مثال، تعداد دفعات قبضه شدن وظایف در سامانه یا تعداد دفعات تجاوز از مهلت مقرر یا بهره‌وری نهایی سامانه.

بر این اساس، نمودار جریان نرمافزار زمان‌بندی در شکل ۳ نمایش داده شده است.



شکل ۳ - نمودار جریان اجرای شبیه‌ساز زمان‌بندی

### ۴ - ۳ ابزارهای استفاده شده

این نرم‌افزار با استفاده از زبان تایپ‌اسکریپت<sup>۱</sup> و در محیط اجرای نود جی‌اس<sup>۲</sup> پیاده‌سازی شده‌است. توسعه‌ی این نرم‌افزار در محیط توسعه‌ی یکپارچه‌ی وب‌استورم<sup>۳</sup> انجام شده‌است و قالب فایل‌های ورودی و خروجی این نرم‌افزار نیز بر پایه‌ی XML<sup>۴</sup> است. برای مدیریت وابستگی‌های پروژه نیز از ابزار یارن<sup>۵</sup> استفاده شده‌است. در ادامه به توضیح مختصر هر کدام از این ابزارها می‌پردازیم.

- **محیط اجرای نود جی‌اس:** اجرای نرم‌افزارهایی که با زبان جاوا اسکریپت<sup>۶</sup> توسعه داده شده‌اند به صورت پیش‌فرض تنها بر روی مرورگرها امکان‌پذیر است و برای اجرای آن‌ها بر روی یک ماشین، به یک محیط اجرا نیازمندیم تا شرایط اجرای نرم‌افزار را فراهم کند. نود جی‌اس یک محیط اجرایی برای جاوا اسکریپت است که بر پایه‌ی موتور جاوا اسکریپت مرورگر کروم ساخته شده است و این امکان را فراهم می‌کند تا منطق برنامه در یک محیط سمت سرور اجرا شود [۸].

- **زبان تایپ‌اسکریپت:** زبان جاوا اسکریپت به صورت پیش‌فرض نوع متغیرها را کنترل نمی‌کند و قوانین مستحکمی در این زمینه ندارد. تایپ‌اسکریپت یک مجموعه‌ی فرآگیر<sup>۷</sup> از زبان جاوا اسکریپت است که قابلیت بررسی و کنترل نوع را به صورت استاتیک فراهم می‌کند و از این نظر کیفیت و قابلیت نگهداری کد را افزایش می‌دهد [۹]. کل این نرم‌افزار با زبان تایپ‌اسکریپت توسعه داده شده که قبل از اجرا به واسطه‌ی دستوری tsc به زبان جاوا اسکریپت تبدیل می‌شود. علل انتخاب زبان تایپ‌اسکریپت برای توسعه‌ی این نرم‌افزار به شرح زیر است:

- قابلیت تعریف، بررسی و کنترل نوع متغیرها
- نحو<sup>۸</sup> سطح بالای ساده و خوانا، خصوصاً در زمینه‌ی مدیریت اشیاء و آرایه‌ها
- پشتیبانی مجتمع از برنامه‌نویسی شی‌گرا و برنامه‌نویسی تابعی

<sup>۱</sup> Typescript

<sup>۲</sup> Node.js Runtime

<sup>۳</sup> Webstorm IDE

<sup>۴</sup> eXtensible Markup Language

<sup>۵</sup> Yarn

<sup>۶</sup> Javascript

<sup>۷</sup> Superset

<sup>۸</sup> Syntax

- پشتیبانی از برنامه‌نویسی غیرهمرونده و رویداد-محور<sup>۱</sup> که شبیه‌سازی دقیق‌تری از رفتار سخت‌افزار فراهم می‌کند
  - دارای کتابخانه‌های متن‌باز گستره و باکیفیت
- **کتابخانه‌ی xml-js:** برای کنترل ورودی و خروجی فایل‌های XML، از کتابخانه xml-js استفاده شده است. این کتابخانه یک راه ساده و راحت برای تبدیل داده‌های XML به اشیاء جاوا اسکریپت و بالعکس فراهم می‌کند [۱۰]، که برای پردازش پیکربندی‌های زمان‌بندی و تولید نتایج ضروری بود. XML، مخفف زبان نشانه‌گذاری قابل توسعه، یک زبان نشانه‌گذاری است که مجموعه‌ای از قوانین را برای کدگذاری اسناد در قالبی تعریف می‌کند که هم برای انسان و هم برای ماشین قابل خواندن است. XML به طور معمول برای ذخیره‌سازی و انتقال داده‌ها، همچنین برای پیکربندی و تعریف ساختار داده‌ها در برنامه‌های مختلف استفاده می‌شود.
- **نرم‌افزار وباستورم:** وباستورم یک محیط توسعه یکپارچه است که توسط شرکت JetBrains به طور خاص برای جاوا اسکریپت و فناوری‌های مرتبط با آن، طراحی شده است. برای تسهیل فرآیند توسعه، از وباستورم با امکانات ویرایشگر کد هوشمند، اشکال‌زدایی و سایر ویژگی‌های خاص آن استفاده شد [۱۱].
- **ابزار مدیریت وابستگی یارن:** یارن یک مدیر بسته (کتابخانه) برای زبان برنامه‌نویسی جاوا اسکریپت است که توسط شرکت‌هایی مانند Google، Facebook، Exponent و Tilde توسعه داده شده است. این ابزار در کنار NPM یکی از مهم‌ترین ابزارهای مدیریت بسته‌های جاوا اسکریپت به شمار می‌رود و برای افزایش سرعت و امنیت در نصب و مدیریت وابستگی‌های پروژه‌های جاوا اسکریپتی به کار می‌رود. Yarn با کش کردن بسته‌های دانلود شده و انجام همزمان عملیات‌ها، سرعت بالایی در نصب و مدیریت بسته‌ها ارائه می‌دهد [۱۲].

در شکل ۴ – تصویری از نماد ابزارهای استفاده شده در این پژوهه ارائه شده است. به ترتیب از چپ به راست: نود جی‌اس، تایپ‌اسکریپت، xml-js، وباستورم و یارن.

---



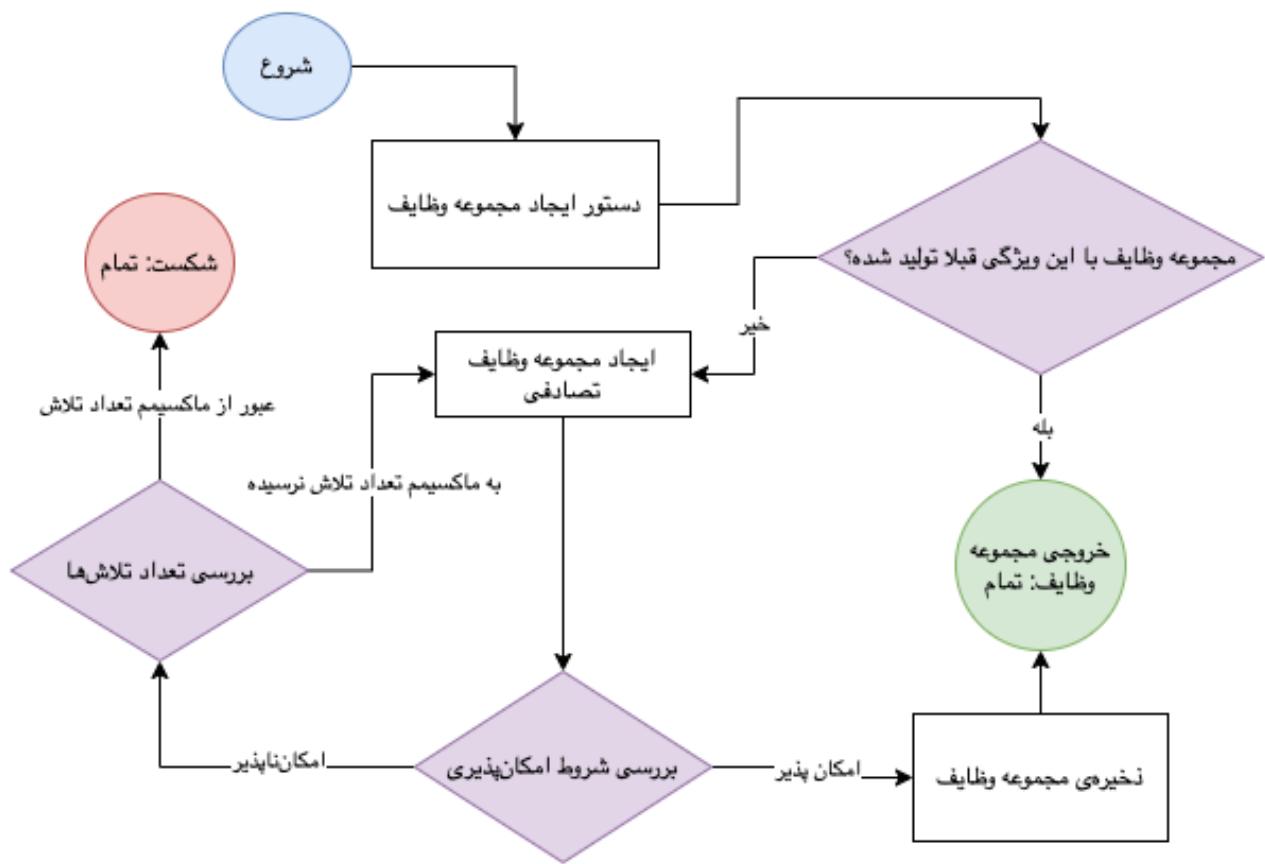
شکل ۴ – نماد ابزارهای استفاده شده برای پیاده‌سازی نرم‌افزار

#### ۴ - ۴ پیاده‌سازی تولید مجموعه وظایف

نرم‌افزار تولید وظایف با رویکرد برنامه‌نویسی تابعی و در یک فایل پیاده‌سازی شده است. ورودی‌ها عبارت‌اند از:

- تعداد وظایف
- بهره‌وری کل مجموعه‌ی وظایف
- ضریب بحرانیت که عددی بزرگتر مساوی یک است و مشخص می‌کند  $C(HI)$  چه مضری از  $C(LO)$  باشد
- احتمال بحرانیت که مشخص می‌کند چه درصدی از کل وظایف، از درجه‌ی بحرانی بالا باشند
- دامنه‌ی تناوب وظایف که به صورت پیش‌فرض بازه‌ی ۱ تا ۱۰۰ در نظر گرفته می‌شود

در خروجی یک فایل XML که وظایفی مطابق با مدل معرفی شده در بخش ۴ - ۱ دارد و شناسه‌ای به صورت ترکیب پنج تایی (تعداد)-(بهره‌وری مجموع)-(ضریب بحرانیت)-(احتمال بحرانیت)-(دامنه تناوب) به آن نسبت داده است، تولید می‌شود. به طور کلی جریان اجرای این بخش از برنامه را می‌توان در نمودار جریان شکل ۵ مشاهده کرد.



شکل ۵- نمودار جریان اجرای نرم‌افزار تولید وظایف

مجموعه وظایف تصادفی با استفاده از روش باروآح و در ۵ مرحله تولید می‌شود [۶]:

۱) توزیع بهره‌وری با استفاده از الگوریتم UUnifast و به دست آوردن بهره‌وری هر وظیفه

۲) توزیع تناوب‌ها با استفاده از الگوریتم log-uniform و به دست آوردن تناوب هر وظیفه

۳) محاسبه‌ی بدترین زمان اجرای خوش‌بینانه هر وظیفه با استفاده از رابطه‌ی

$C(HI) = CF \times C(LO)$  (محاسبه‌ی بدترین زمان اجرای بدینانه هر وظیفه با استفاده از ضریب بحرانیت)

۴) تخصیص درجه‌ی بحرانیت هر وظیفه به صورت تصادفی با استفاده از احتمال بحرانیت

بعد از آن بررسی می‌شود که مجموعه‌ی تولیدشده امکان‌پذیر باشد و به این منظور روابط  $C(HI) \leq C(LO)$  و  $T < C(HI)$  باید برای هر وظیفه برقرار باشند. در صورتی که بعد از ۵۰۰۰۰ تلاش، این روابط برای مجموعه‌ی وظایف

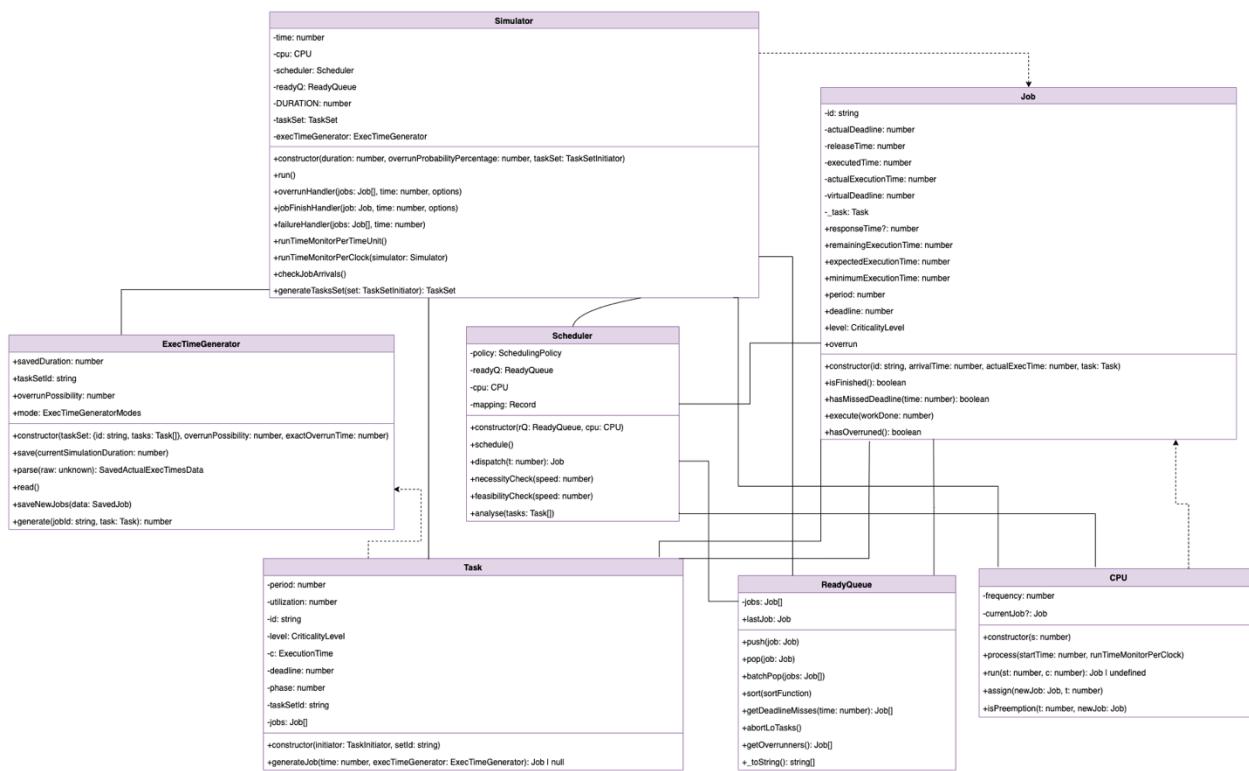
برقرار نباشند، برنامه با خطا به پایان می‌رسد. در سناریوی تولید موفقیت‌آمیز، مجموعه وظایف بعد از ذخیره، برای استفاده‌ی شبیه‌ساز زمان‌بندی فرمت می‌شود.

تابع `UUniFast` الگوریتم `uunifast` را پیاده‌سازی می‌کند تا بهره‌وری کل را بین تعداد داده‌شده‌ای از وظایف توزیع کند، به طوری که جمع بهره‌وری‌ها با کل مشخص شده مطابقت داشته باشد. این الگوریتم برای ایجاد مجموعه‌های وظایف واقع‌بینانه که محدودیت‌های سامانه‌های بی‌درنگ را رعایت می‌کنند حیاتی است، چرا که کل بهره‌وری نمی‌تواند از ظرفیت سامانه فراتر رود. الگوریتم `UUniFast` این را با تقسیم تکراری کل بهره‌وری به بخش‌های کوچکتر و اختصاص هر بخش کوچک به یک وظیفه، در حالی که جمع کلی حفظ می‌شود، به دست می‌آورد.

تابع `logUniform` برای تولید تناوب وظایف با استفاده از توزیع لگاریتم-یکنواخت به کار می‌رود که طیف متنوعی از تناوب‌ها را برای وظایف فراهم می‌کند. این توزیع از تناوب وظایف واقع‌بینانه‌تر است چرا که در سامانه‌های دنیای واقعی، تناوب‌ها می‌توانند بر روی چندین مرتبه از اندازه متفاوت باشند. استفاده از مقیاس لگاریتمی اطمینان می‌دهد که تناوب‌های تولیدی به طور یکنواخت در سراسر دامنه مشخص شده پخش می‌شوند، که امکان بررسی جامع‌تر سناریوهای زمان‌بندی مختلف را فراهم می‌آورد.

## ۴ - ۵ پیاده‌سازی شبیه‌سازی زمان‌بندی

نرم‌افزار شبیه‌سازی زمان‌بندی با رویکرد برنامه‌نویسی شی‌گرا و در چندین کلاس مجزا پیاده‌سازی شده است. نمودار کلاس این نرم‌افزار در شکل ۶ نشان داده شده است. در این بخش به توضیح هر کدام از واحدهای تشکیل‌دهنده‌ی نرم‌افزار می‌پردازیم و مولفه‌های سازنده‌ی هر کدام را به همراه عملکرد و جزئیات پیاده‌سازی شرح می‌دهیم. پس از آشنایی و شناخت این واحدهای تشکیل‌دهنده، به توضیح ارتباط بین آن‌ها می‌پردازیم.



شکل ۶ – نمودار کلاس‌های نرم‌افزار شبیه‌سازی

## ۴ - ۵ - ۱ متغیرهای سراسری

در این شبیه‌سازی دو متغیر سراسری از جنس شیء با نام‌های System و Config تعریف شده‌اند که به ترتیب هر کدام نگهدارنده‌ی تنظیمات زمان‌بندی و ویژگی‌های اصلی سامانه شبیه‌سازی شده هستند. هر کدام از کلاس‌های نرم‌افزار به متغیرهای SystemConfig و Config دسترسی دارند و می‌توانند مقادیر آن‌ها را بخوانند.

## ۴ - ۵ - ۱ - ۱ سامانه شبیه‌سازی

شیء System که ویژگی‌های اصلی سامانه شبیه‌سازی شده را نگهداری می‌کند، از دو مولفه‌ی level و virtualDeadlineFactor تشکیل شده است. مقدار مولفه‌های System در طول اجرای برنامه قابل تغییر است. توضیح این مولفه‌ها به شرح زیر است:

- این مولفه با مقادیر LO یا HI مشخص می‌کند که در هر لحظه از اجرا، سامانه در چه درجه‌ای از بحرانیت قرار دارد. در اصل، این مولفه نماینده‌ی مقدار Γ یا درجه‌ی بحرانیت سامانه است.

virtualDeadlineFactor: این مولفه‌ی عددی نشان‌دهنده‌ی مضرب مهلت مجازی (مخفف شده به صورت vdf) در زمان‌بندی سامانه با الگوریتم EDF-VD که در بخش ۳ - ۳ - ۲ توضیح داده شد، می‌باشد و از رابطه‌ی  $vdf = \frac{U_2(1)}{1-U_1(1)}$  به دست می‌آید و به صورت  $virtualDeadline_{ij} = vdf * D_i + r_{ij}$  استفاده می‌شود. در حالتی که سامانه با الگوریتم EDF یا در حالت سنتی (تک بحرانیتی) زمان‌بندی می‌شود، مقدار این مولفه ۱ است.

#### ۴ - ۱ - ۵ - تنظیمات زمان‌بندی

شیء Config نگهدارنده‌ی تنظیمات اصلی زمان‌بندی است و از ۵ مولفه‌ی اصلی تشکیل شده‌است. این متغیر در ابتدای شروع شبیه‌سازی قفل می‌شود و مقدار آن در سراسر اجرای برنامه ثابت خواهد ماند، بنابراین مقداردهی به آن، تنها قبل از شروع اجرا امکان‌پذیر است. مولفه‌های تشکیل‌دهنده‌ی شیء Config عبارت‌اند از:

exactOverrunTime: یک مولفه با مقدار عددی طبیعی است که زمان رخداد تجاوز از زمان اجرا را در سامانه شبیه‌سازی می‌کند. در صورتی که این مولفه مقداری بزرگتری از صفر داشته باشد و سامانه نیز شرایط افزایش درجه‌ی بحرانیت را داشته باشد، بدون توجه به اینکه تجاوز از زمان اجرا در کارهای موجود در سامانه اتفاق افتاده است یا خیر، در لحظه‌ی exactOverrunTime سامانه از حالت LO به حالت HI می‌رود. کاربرد این مولفه برای شبیه‌سازی رخداد تجاوز از زمان اجرا به صورت تعامدی و در لحظه‌ای خاص است.

traditional: مولفه‌ای از جنس بولی ۱ است که در حالت درست، سامانه را به صورت یک سامانه سنتی یعنی یک سامانه بی‌درنگ ساده با یک درجه‌ی بحرانیت ثابت شبیه‌سازی می‌کند. در چنین شرایطی، سامانه امکان تغییر درجه‌ی بحرانیت را ندارد و رخداد تجاوز از زمان اجرا نظارت نمی‌شود. کاربرد این مولفه برای مقایسه‌ی زمان‌بندی‌پذیری یک مجموعه وظایف بحرانی مختلط در حالت سنتی و در حالت بحرانی مختلط است.

workDonePerClock: مولفه‌ای با دو مقدار عددی ۰.۱ یا ۱ است و مشخص می‌کند که در هر تیک از پردازنده، چه مقدار کار انجام می‌شود. اگر این مولفه با مقدار ۰.۱ تنظیم شود، نرم‌افزار دقیقی برابر با ۰.۱ خواهد داشت و در حالتی که این مولفه با مقدار ۱ تنظیم شود، اعداد موجود در نرم‌افزار باید صحیح باشند؛ به عبارت دیگر، مجموعه وظایف سامانه باید با مقادیر صحیح تعریف شوند.

<sup>۱</sup> Boolean

- مولفه‌ای با مقادیر عددی طبیعی است که نشان می‌دهد در هر واحد زمانی، پردازنده چند تیک می‌زند. کل کار پردازش شده در یک واحد زمانی از رابطه‌ی  $\text{workDonePerClock} \times \text{frequency}$  محاسبه می‌شود؛ برای مثال، در صورتی که  $\text{frequency}$  برابر با ۰.۱ و  $\text{workDonePerClock}$  برابر با ۲۰ باشد، پردازنده در هر واحد زمانی می‌تواند به اندازه‌ی ۲ واحد کار را پردازش کند.
- مولفه‌ای با مقادیر HI یا LO می‌باشد که مشخص می‌کند سامانه با چه درجه‌ای از بحرانیت شروع به کار می‌کند. در ابتدای شروع شبیه‌سازی مولفه level از متغیر سامانه برابر با initialSystemLevel تنظیم خواهد شد.

ترکیب متغیرهای سامانه و تنظیمات و مولفه‌های گسترده‌ی آن‌ها به کاربر اجازه می‌دهد که یک مجموعه‌ی وظایف را در حالت‌های مختلف زمان‌بندی کند و زمان‌بندی‌پذیری آن را در حالت‌ها مختلف با هم مقایسه کند.

## ۴ - ۵ - ۲ کلاس شبیه‌ساز

کلاس شبیه‌ساز با نام Simulator، واحد اصلی اجرای نرم‌افزار شبیه‌سازی می‌باشد. اجرای نرم‌افزار شبیه‌سازی با ایجاد یک نمونه از این کلاس و فراخوانی تابع run بر روی آن اتفاق می‌افتد. ورودی‌های موردنیاز برای ایجاد کلاس شبیه‌ساز، داده‌های مربوط به ایجاد مجموعه وظایف بحرانی مختلط، مدت زمان اجرا و احتمال تجاوز کارها از بدترین زمان اجرای آن‌ها می‌باشد. وقتی یک نمونه از کلاس شبیه‌ساز ایجاد می‌شود، این نمونه با استفاده از داده‌های مربوط به ایجاد مجموعه وظایف بحرانی مختلط، با استفاده از تابع generateTasksSet، وظایف سامانه را که از جنس کلاس وظیفه هستند در سامانه ایجاد کرده و در متغیر taskset نگهداری می‌کند. در تابع run حلقه‌ی اصلی سامانه اجرا می‌شود و در این حلقه زمان سامانه با قدم‌هایی به اندازه‌ی ۱ جلو می‌رود.

متغیرهای کلاس شبیه‌ساز به شرح زیر هستند:

- متغیری از جنس عدد است و زمان سامانه شبیه‌سازی شده را نگهداری می‌کند.
- یک نمونه از کلاس پردازنده می‌باشد و پردازنده‌ی اصلی سامانه است که وظایف بر روی آن زمان‌بندی و اجرا می‌شوند.
- یک نمونه از کلاس صفات آماده است و کارهای آماده‌ی اجرا در سامانه را نگهداری می‌کند.
- یک نمونه از کلاس زمان‌بند است که وظیفه‌ی زمان‌بندی کارهای فعال در سامانه و تخصیص وظیفه به cpu را بر عهده دارد. همینطور بررسی‌های اولیه‌ی مربوط به زمان‌بندی را انجام می‌دهد.
- متغیری از جنس عدد است که مدت زمان اجرای شبیه‌سازی در آن نگهداری می‌شود.

taskset: یک شی متشکل از دو بخش id و tasks است. id یک متغیر از نوع رشته حروف می‌باشد که شناسه‌ی یکتای مجموعه‌ی وظایف در حال اجرا در سامانه است و tasks یک آرایه از وظایفی است که در این شبیه‌سازی زمان‌بندی می‌شوند.

execTimeGenerator: یک نمونه از کلاس تولید زمان اجرای حقیقی است. این کلاس زمان اجرای حقیقی کارهای سامانه را به صورت تصادفی تولید می‌کند.

تابع کلاس شبیه‌ساز به شرح زیر هستند:

constructor: کلاس شبیه‌ساز را با استفاده از یک مدت زمان مشخص، احتمال تجاوز زمانی و داده‌های تولید مجموعه‌ای از وظایف، مقداردهی اولیه می‌کند. سپس با فراخوانی تابع generateTasksSet داده‌های تولید مجموعه‌ای از وظایف را به نمونه‌هایی از کلاس وظیفه تبدیل می‌کند.

run: وظیفه‌ی اجرای حلقه‌ی اصلی شبیه‌سازی را به عهده دارد.

overrunHandler: در صورت وقوع تجاوز از بدترین زمان اجرای وظایف، این رخداد را مدیریت می‌کند و در صورت امکان، با حذف وظایف از درجه‌ی بحرانی پایین از صف کارهای آماده، درجه‌ی بحرانیت سامانه را به HI تغییر می‌دهد.

jobFinishHandler: در صورت اتمام اجرای یک کار، این رخداد را مدیریت می‌کند و با استفاده از کلاس زمان‌بند، وظیفه‌ی جدیدی را به پردازنده اختصاص می‌دهد.

failureHandler: در صورتی که سامانه به دلیل تجاوز از مهلت واقعی وظایف از درجه‌ی بحرانی بالا، با خطا روبرو شود، این خطا را مدیریت کرده و شبیه‌سازی را به پایان می‌رساند.

runTimeMonitorPerTimeUnit: بعد از هر قدم از اجرای حلقه‌ی اصلی شبیه‌سازی و به ازای هر واحد زمانی، برای بررسی تجاوز از زمان اجرا و از دست رفتن مهلت‌ها، سامانه را نظارت می‌کند و در صورت لزوم تابع failureHandler را فراخوانی می‌کند.

runTimeMonitorPerClock: بعد از هر تیک ساعت که در پردازنده اجرا می‌شود، برای بررسی تجاوز از زمان اجرا و اتمام وظایف، بر سامانه نظارت می‌کند. به ازای هر بار فراخوانی پردازنده، یک نمونه از این تابع به پردازنده انتقال داده می‌شود. در صورتی که اجرای وظیفه کامل شده باشد یا تجاوز از زمان اجرا اتفاق افتاده باشد، این تابع jobFinishHandler یا overrunHandler را فراخوانی می‌کند.

checkJobArrivals: به ازای هر واحد زمانی از سامانه بررسی می‌کند که وظایف موجود در سامانه، کار جدیدی را منتشر کرده‌اند یا نه و در صورت لزوم صف کارهای آماده را بهروزرسانی کرده و با استفاده از کلاس زمان‌بند، کار جدیدی را برای اجرا شدن به پردازنده اختصاص می‌دهد.

- generateTaskSet: مجموعه‌ای از وظایف را که نمونه‌هایی از کلاس وظیفه هستند، بر اساس داده‌های ورودی شبیه‌سازی تولید می‌کند و در متغیر taskset می‌ریزد.

### ۴ - ۵ - ۳ کلاس وظیفه

هدف کلاس وظیفه با نام Task، تولید کارها یا به تعبیری واحدهای اجرایی سامانه است. هر وظیفه با استفاده از متغیر تناوب (period) از جنس عدد طبیعی، متغیر درجه‌ی بحرانیت (level) با مقدار LO یا HI یا مقدار مهلت اجرا (deadline) از جنس عدد گویا، متغیر فاز (phase) از جنس عدد طبیعی و متغیر بدترین زمان‌های اجرا (c) تعریف می‌شود. متغیر c از جنس شی‌ای متشکل از دو بخش LO و HI است که به ترتیب نشان‌دهنده‌ی بدترین زمان‌های اجرای خوش‌بینانه و بدینانه- به عبارتی بدترین زمان اجرا در حالت LO و بدترین زمان اجرا در حالت HI- و از جنس عدد گویا هستند. علاوه‌بر این‌ها، کلاس وظیفه متغیری به نام jobs دارد که آرایه‌ای از تمام کارهای منتشرشده توسط این وظیفه می‌باشد.

مهم‌ترین تابع این کلاس generateJob است که زمان سامانه را به عنوان ورودی دریافت می‌کند و در صورتی که رابطه‌ی  $time = k \times period + phase$  (یک عدد صحیح است) برقرار باشد، یک نمونه از کلاس کار ایجاد می‌کند، آن را به آرایه‌ی jobs می‌افزاید و سپس به کلاس شبیه‌ساز باز می‌گرداند تا به صفت کارهای آماده اضافه شود. این تابع برای ساختن کارها از یک نمونه از کلاس تولید زمان اجرای حقیقی استفاده می‌کند و تابع generate از این کلاس را فراخوانی می‌کند تا یک زمان اجرای واقعی را به صورت تصادفی برای کار منتشرشده تولید کند.

### ۴ - ۵ - ۴ کلاس کار

کلاس کار با نام Job، یکی از اصلی‌ترین واحدهای تشکیل‌دهنده‌ی نرم‌افزار شبیه‌سازی است. این کلاس بعد از کلاس شبیه‌ساز، مهم‌ترین کلاس نرم‌افزار است. کلاس کار از متغیرها و توابعی تشکیل شده‌است که می‌توانند وضعیت فعلی یک کار را مشخص کنند و از این نظر، کلاس کار یک کلاس خودکفاست. علاوه بر این برای جلوگیری از ایجاد متغیرهای تکراری، کلاس کار به یک نمونه از کلاس وظیفه‌ای که آن را ایجاد کرده است دسترسی دارد و مقادیری مانند تناوب و درجه‌ی بحرانیت خود را از این کلاس وظیفه دریافت می‌کند.

متغیرهای کلاس کار به شرح زیر هستند:

- id: شناسه‌ی یکتای یک کار که از ترکیب شناسه‌ی وظیفه‌ی منتب به یک کار و شماره‌ی آن کار تشکیل شده است. اگر رابطه‌ی  $t = j \times T_i + \varphi_i$  برای یک کار بقرار باشد، شناسه‌ی آن کار یک رشته به صورت  $j-i$ -خواهد بود.
- releaseTime: متغیری از جنس عدد و با دقت ۱.۰ است که مقدار زمان انتشار یک کار یا  $r_{ij}$  را در خود نگهداری می‌کند و در زمان ایجاد یک نمونه از کلاس کار، مقداردهی می‌شود.
- actualDeadline: متغیری از جنس عدد است که مهلت واقعی یک کار را درون خود نگهداری می‌کند. مهلت واقعی یک کار از رابطه‌ی  $deadline_{ij} = D_i + r_{ij}$  به دست می‌آید.
- virtualDeadline: متغیری از جنس عدد است که مهلت مجازی یک کار را درون خود نگهداری می‌کند که این مقدار از رابطه‌ی  $virtualDeadline_{ij} = vdf \times D_i + r_{ij}$  به دست می‌آید. vdf یک عدد است که در حالت زمان‌بندی سامانه با الگوریتم EDF-VD در مهلت نسبی هر وظیفه از درجه‌ی بحرانی بالا ضرب می‌شود و مقدار آن از مولفه‌ی  $virtualDeadlineFactor$  از متغیر سراسری سامانه خوانده می‌شود. در صورتی که وظیفه از درجه‌ی بحرانی پایین باشد یا سامانه با الگوریتم EDF زمان‌بندی شود، مقادیر مهلت واقعی و مجازی یک کار با هم برابر خواهند بود.
- executedTime: متغیری از جنس عدد است که مشخص می‌کند یک کار تا به این لحظه، چه مقداری از ظرفیت پردازندۀ را به خود اختصاص داده است یا به عبارت دیگر چقدر اجرا شده است.
- actualExecutionTime: متغیری از جنس عدد است که مشخص می‌کند زمان اجرای حقیقی یک کار چقدر است. این مقدار به صورت تصادفی توسط کلاس تولید زمان اجرای حقیقی تولید می‌شود و سازوکار تولید آن در بخش‌های بعدی توضیح داده خواهد شد.
- task: یک شیء از کلاس وظیفه است و به وظیفه‌ای که کار را ایجاد کرده است اشاره می‌کند.
- responseTime: متغیری از جنس عدد است که زمان پاسخگویی یک کار در آن نگهداری می‌شود.
- executedAtLevel: متغیری با مقادیر HI یا LO است که نشان می‌دهد یک وظیفه در چه درجه‌ی بحرانیتی از سامانه اجرای خود را کامل کرده است.

توابع کلاس کار به دو دسته‌ی توابع get و توابع عملکردی تقسیم می‌شوند. توابع get در اصل یکی از متغیرهای تعریف‌کننده‌ی کار را با شرایطی خاص محاسبه کرده و باز می‌گردانند. در ادامه به شرح توابع کلاس کار می‌پردازیم:

- get remainingExecutionTime: این تابع مقدار زمان باقی‌مانده از اجرای یک کار را از تفاضل مقادیر متغیرهای  $actualExecutionTime$  از  $executedTime$  به دست می‌آورد. در اصل مشخص می‌کند که یک کار چقدر دیگر باید اجرا شود.

- get expectedExecutionTime: این تابع با توجه به درجهی بحرانیت سامانه مشخص می‌کند که مقدار WCET یک کار چقدر است. در صورتی که سامانه در حالت HI باشد مقدار بازگشتی تابع برابر است با  $\tilde{C}_i(HI)$  و در صورتی که سامانه در حالت LO باشد، تابع مقدار  $(LO)_i$  را بازمی‌گرداند.
- get minimumExecutionTime: تابع در هر حالتی مقدار  $(LO)_i$  را باز می‌گرداند.
- get period: این تابع مقدار تناوب یک کار را با توجه به وظیفه‌ی منتبه به آن کار محاسبه کرده و باز می‌گرداند.
- get deadline: با توجه به اینکه سامانه تغییر حالت داده‌است یا نه، این تابع یکی از مقادیر virtualDeadline یا actualDeadline را به عنوان مهلت یک کار در لحظه‌ی فراخوانی باز می‌گرداند.
- get level: یک تابع درجهی بحرانیت یک کار را از روی وظیفه‌ی منتبه به آن به دست می‌آورد و باز می‌گرداند.
- get overrun: این تابع محاسبه می‌کند که یک کار، از زمان اجرای موردنظر برای آن چقدر فراتر اجرا شده‌است و این مقدار عددی را باز می‌گرداند.
- get remainingExecutionTime: این تابع با درنظر گرفتن مقدار بازگشتی از تابع isFinished مشخص می‌کند که یک کار اجرای خود را کامل کرده‌است یا خیر و در صورتی که اجرای کار کامل شده باشد، مقدار responseTime را محاسبه می‌کند.
- hasMissedDeadline: این تابع با دریافت متغیر time به عنوان ورودی، فراخوانی می‌شود و در هر لحظه مشخص می‌کند که مهلت زمانی یک کار از دست رفته‌است یا خیر؛ به عبارتی، کار از مهلت زمانی خود تجاوز کرده‌است یا خیر.
- hasOverrunned: این تابع مقادیر executedTime و expectedExecutionTime را با هم مقایسه می‌کند و در صورتی که کار از زمان اجرای منتبه به آن در درجهی بحرانیت فعلی سامانه فراتر رفته باشد، این تجاوز از زمان اجرا را اعلام می‌کند.
- execute: این تابع با دریافت متغیر عددی workDone فراخوانی می‌شود و مقدار executedTime یک کار را به اندازه‌ی workdone جلو می‌برد. این تابع توسط پردازنده بر روی کار منتبه به آن فراخوانی می‌شود و هدف آن شبیه‌سازی اجرای یک کار توسط پردازنده است.

ترکیب این توابع و مقادیر با عملکردها و تعاریف گسترده و دقیقی که فراهم می‌کنند، کلاس کار را به یک کلاس خودکفا تبدیل می‌کند. به این ترتیب در هر لحظه از اجرای شبیه‌سازی یا هر نقطه از پیاده‌سازی سامانه شبیه‌ساز، با دسترسی به یک نمونه کار، می‌توان وضعیت آن را به راحتی ارزیابی کرد و درباره‌ی سامانه شبیه‌سازی شده

تصمیم گرفت؛ برای مثال، بعد از هر تیک از پردازنده، با فراخوانی تابع hasOverruned روی کاری که در آن تیک توسط پردازنده اجرا شده است، می‌توان سنجید که سامانه باید وارد حالت عملیاتی بحرانی شود یا آنکه می‌تواند در حالت عادی به اجرای خود ادامه دهد.

#### ۴ - ۵ - کلاس پردازنده

کلاس پردازنده با نام CPU، واحد پردازشی نرم‌افزار زمان‌بندی سامانه‌های بحرانی مختلط را شبیه‌سازی می‌کند. در زمان شروع شبیه‌سازی، یک نمونه از این کلاس در کلاس شبیه‌ساز ساخته شده و در متغیر cpu نگهداری می‌شود. این کلاس از چهار بخش اساسی تشکیل شده است:

- متغیر frequency: متغیری از جنس عدد است که مشخص می‌کند پردازنده در هر واحد زمان، یا به عبارت دیگر در هر بار فراخوانی تابع process، چند تیک می‌زند. این متغیر با استفاده از مولفه frequency از متغیر سراسری تنظیمات مقداردهی اولیه می‌شود..
- متغیر currentJob: یک نمونه از کلاس کار است که پردازنده در هر تیک آن را با فراخوانی تابع execute به اندازه‌ی مشخص شده در مولفه‌ی workDonePerClock از متغیر سراسری تنظیمات، اجرا می‌کند. این کار توسط کلاس زمان‌بند به پردازنده تخصیص داده می‌شود.

تابع process: این تابع حلقه‌ی درونی برنامه را که پیش‌تر در بخش ۴ - ۲ به آن اشاره شد، اجرا می‌کند. عملکرد اصلی این تابع، شبیه‌سازی تیک پردازنده است و در هر واحد زمانی، یک بار فراخوانی می‌شود. حلقه‌ی پیاده‌سازی‌شده توسط این تابع از تیک صفر تا تیک = فرکانس پردازنده، با قدمهایی به اندازه‌ی یک پیش‌می‌رود و در هر دور، یک پردازش به اندازه‌ی currentJob workDonePerClock روی runTimeMonitorPerClock انجام می‌دهد و پس از هر دور، تابع currentJob انجام بدهد.

- تابع isPreemption: این تابع وظیفه دارد تا در صورت تغییر مقدار currentJob، در زمانی که زمان‌بند یک کار جدید را به پردازنده اختصاص می‌دهد، تشخیص بدهد که قبضه صورت گرفته است یا خیر و در صورت وقوع این رخداد، آن را با استفاده از کلاس ذخیره‌کننده‌ی Rخدادهای اجرا، ذخیره کند.

## ۴ - ۵ - ۶ کلاس زمان‌بند

کلاس زمان‌بند با نام Scheduler، وظیفه‌ی زمان‌بندی و بررسی‌های اولیه‌ی زمان‌بندی را به عهده دارد. در ابتدای شبیه‌سازی و قبل از شروع انتشار کارها و مراحل پردازشی، تابع analyse فراخوانی می‌شود و بررسی‌های بهره‌وری-محور بر روی مجموعه‌ی وظایف انجام می‌شود. این بررسی‌ها بر اساس الگوریتم EDF-VD پیاده‌سازی شده‌اند که پیش‌تر در بخش ۳ - ۳ - ۲ توضیح داده شد. در صورتی که مجموعه‌ی وظایف تنها با استفاده از مهلت‌های مجازی قابل زمان‌بندی باشد، مولفه‌ی virtualDeadlineFactor از متغیر سراسری سامانه مقداردهی می‌شود تا مهلت‌های مجازی کارهای منتشر شده، توسط این مقدار محاسبه شوند. نتایج این بررسی‌ها با استفاده از کلاس ذخیره‌کننده‌ی Rخدادهای اجرا، ذخیره می‌شوند.

تنها نمونه‌ی ایجاد شده از کلاس صف کارهای آماده، توسط کلاس شبیه‌ساز به کلاس زمان‌بند انتقال داده می‌شود و کلاس زمان‌بند در هنگام فراخوانی تابع schedule، این صف را بر اساس متغیر deadline از هر کار که سیاست‌های محاسبه‌ی آن در بخش ۴ - ۵ - ۴ توضیح داده شد، به صورت صعودی مرتب‌سازی می‌کند. البته این امکان وجود دارد که کلاس زمان‌بند برای اعمال انواع سیاست‌های زمان‌بندی گسترش داده شود اما تمرکز این پیاده‌سازی الگوریتم EDF-VD بوده است. در صورت فراخوانی تابع dispatch، زمان‌بند اولین کار از صف کارهای آماده‌ی مرتب‌شده را برای اجرا به پردازنده اختصاص می‌دهد و این رخداد را به همراه زمان وقوع آن ذخیره می‌کند تا در انتهای شبیه‌سازی بتواند یک برنامه‌ی زمان‌بندی کامل را به عنوان خروجی تولید کند.

## ۴ - ۵ - ۷ کلاس صف کارهای آماده

کلاس صف کارهای آماده با نام ReadyQueue، کارهای منتشرشده در سامانه را نگهداری می‌کند و توابعی را فراهم می‌کند که با استفاده از آن‌ها می‌توان صف کارها را مدیریت کرد. این تابع یک آرایه از کارهای منتشر شده در سامانه را نگهداری می‌کند و توابع آن مدیریت این آرایه را تسهیل می‌کنند. تابع push یک کار جدید را به این آرایه اضافه می‌کند. تابع pop و batchPop به ترتیب یک کار و آرایه‌ای از کارها را حذف می‌کنند. تابع sort بر اساس سیاستی که به عنوان ورودی دریافت می‌کند، آرایه را مرتب‌سازی می‌کند. تابع lastJob اولین کار آرایه را که کاندید اجرا بر روی پردازنده است باز می‌گرداند. تابع getDeadlineMisses آرایه‌ی کارها را می‌پیماید و تمام کارهایی را که تابع hasMissedDeadline آن‌ها مقدار درست باز می‌گرداند، جداسازی می‌کند. تابع abortLoTasks تمامی کارهایی که از درجه‌ی بحرانی L0 هستند را از آرایه‌ی کارها حذف می‌کند. تابع getOverruners نیز تمامی کارهایی را که تابع hasOverruned آن‌ها مقدار درست باز می‌گرداند، جداسازی

می‌کند. به این ترتیب، صفت کارهای آماده تنها یک آرایه است که قابلیت‌های آن برای یک نرم‌افزار شبیه‌سازی زمان‌بندی گسترش داده شده است.

#### ۴ - ۵ - ۸ کلاس تولید زمان اجرای حقیقی

کلاس تولید زمان اجرای حقیقی با نام ExecTimeGenerator، یکی از کلاس‌های مهم این نرم‌افزار است که می‌تواند در چهار حالت عملیاتی، زمان اجرای حقیقی کارها را تولید کند. در دنیای واقعی، تنها اطلاعات موجود از زمان اجرای یک وظیفه، مجموعه WCET آن وظیفه است و تا زمانی که کارهای منتشرشده از یک وظیفه در سامانه اجرا نشوند، نمی‌توان زمان اجرای حقیقی کارها را به طور قطعی به دست آورد. عواملی از جمله انتظار برای I/O یا زمان پاسخگویی کش و حافظه اصلی، می‌توانند در زمان اجرای یک وظیفه تغییر ایجاد کنند. از آن جایی که نرم‌افزار ما تنها یک نرم‌افزار شبیه‌سازی زمان‌بندی است و شامل پیچیدگی‌های I/O و حافظه نمی‌شود، باید سازوکاری در آن تعییه شود که بتواند زمان اجرای حقیقی یک کار را شبیه‌سازی کند؛ کلاس تولید زمان اجرای حقیقی، وظیفه‌ی پیاده‌سازی این سازوکار را بر عهده دارد. به این ترتیب، در زمان انتشار هر کار، کلاس وظیفه از کلاس تولید زمان اجرای حقیقی می‌خواهد که زمان واقعی اجرا یا actualExecutionTime آن کار را محاسبه کند.

کلاس تولید زمان اجرای حقیقی در زمان ایجاد، یک متغیر عددی به نام overrunPossibility و یک حالت عملیاتی را از کلاس شبیه‌ساز به عنوان ورودی دریافت می‌کند و زمان اجرای واقعی کارها را با توجه به این مقادیر، به صورت تصادفی یا به صورت ثابت تولید می‌کند. در حالت ثابت، زمان اجرای حقیقی یک وظیفه می‌تواند برابر با یکی از بدترین زمان‌های اجراییش باشد. در حالت تصادفی، زمان اجرای حقیقی وظیفه با استفاده از overrunPossibility تولید می‌شود. در صورتی که overrunPossibility برابر ۲۰ باشد، هر زمان اجرای واقعی تولیدشده با احتمال ۲۰ درصد از بدترین زمان اجرای خوش‌بینانه آن کار بزرگتر خواهد بود؛ به عبارت دیگر، این کار در زمان اجرا حتماً باعث وقوع رخداد تجاوز از زمان اجرا خواهد شد. با استفاده از این سازوکار می‌توان وقوع تجاوز از زمان اجرا را به صورت تصادفی در این نرم‌افزار شبیه‌سازی کرد.

برای حالت عملیاتی تصادفی، از آن جایی که در هر بار اجرا ممکن است اعداد جدیدی تولید شوند، و از سوی دیگر کاربران نیاز دارند که یک مجموعه وظایف یکسان با مقادیر ثابت را با تنظیمات متفاوت اجرا کنند، کلاس تولید زمان اجرای حقیقی، زمان‌های تصادفی تولیدشده برای هر کار را در یک فایل xml ذخیره و نگهداری می‌کند تا در اجراهای متفاوت، زمان‌های یکسانی را به کاربران عرضه کند. در صورتی که شناسه‌ی مجموعه وظایف یا مقدار overrunPossibility تغییر کند یا فایل xml ذخیره‌شده حذف شود، زمان‌های تصادفی جدیدی تولید خواهند شد.

## ۴ - ۵ - ۹ کلاس ذخیره‌کننده‌ی رخدادهای اجرا

کلاس ذخیره‌کننده‌ی رخدادهای اجرا با نام `Logger`، وظیفه‌ی ذخیره کردن<sup>۱</sup> تمام رخدادهای شبیه‌سازی، نمایش آن‌ها در حین اجرای برنامه و تولید فایل‌های خروجی را دارد. این کلاس از بخش‌های اصلی سامانه زمان‌بندی نمی‌باشد، اما وجود آن برای تولید خروجی‌های خوانا ضروری است. یک نمونه از این کلاس در ابتدای اجرای نرم‌افزار ایجاد می‌شود و به صورت سراسری در اختیار تمام کلاس‌های دیگر قرار می‌گیرد. در جدول ۶، تمامی رخدادهایی که توسط این کلاس، ذخیره و پردازش می‌شوند نشان داده شده است. یکی از ویژگی‌های مهم این کلاس، قابلیت تنظیم نمایش هر رخداد است؛ به طور مثال، می‌توان تعیین کرد که رخدادهای `clock` و `utilization` در حین اجرای شبیه‌سازی به کاربر نمایش داده نشود یا نمایش کل رخدادها را به صورت یکجا خاموش کرد.

در پایان پردازش، این کلاس با توجه به اطلاعاتی که از تمام رخدادهای سامانه در اختیار دارد، محاسباتی آماری از زمان‌بندی انجام‌شده ارائه می‌دهد و به همراه زمان‌بندی نهایی در فایل خروجی ذخیره می‌کند. این اطلاعات آماری شامل نتایج آزمون‌های زمان‌بندی، بهره‌وری وظایف از درجه‌ی بحرانی متفاوت، تعداد قبضه‌ها، تعداد تجاوز از مهلت اجرا، تعداد کل کارهای منتشر شده، تعداد کل کارهای کامل شده، کمترین، بیشتری و میانگین زمان پاسخگویی هر وظیفه و لحظه‌ی تغییر حالت سامانه به همراه علت آن است.

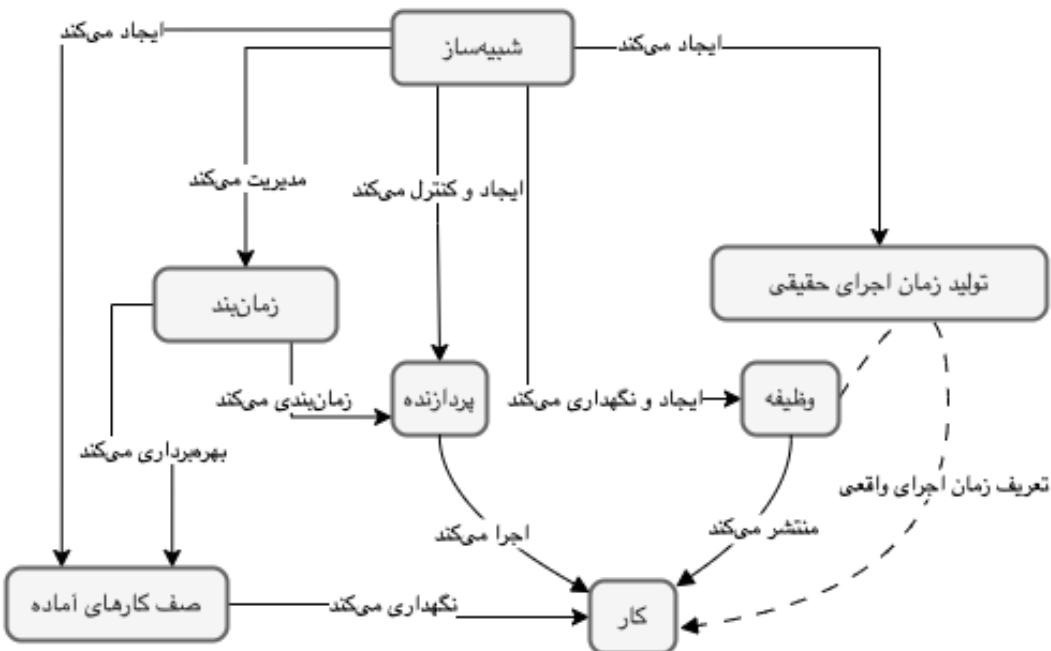
جدول ۶ - رخدادهای ذخیره‌شده در حین اجرای شبیه‌سازی

توضیحات	نام رخداد	شناسه رخداد
قبل از شروع زمان‌بندی، زمانی که تابع <code>analyse</code> از کلاس زمان‌بند فراخوانی می‌شود، میزان بهره‌وری مجموعه‌ی وظایف برای بررسی زمان‌بندی پذیری محاسبه می‌شود. نتایج این محاسبات ذخیره می‌شوند.	بهره‌وری	<code>utilization</code>
در ابتدای هر واحد زمانی، انتشار کارهای جدید بررسی می‌شود. در صورتی که کار جدیدی منتشر شود، اطلاعات آن ذخیره می‌شود.	انتشار کار جدید	<code>arrival</code>
زمانی که تابع <code>analyse</code> از کلاس زمان‌بند فراخوانی می‌شود، با استفاده از تست‌های بهره‌وری-محور، امکان‌پذیری مجموعه‌ی وظایف و زمان‌بندی پذیر مجموعه وظایف با الگوریتم EDF-VD را بررسی می‌کند. نتایج این بررسی، ذخیره می‌شوند.	آزمون امکان‌پذیری	<code>feasibilityTest</code>
در لحظه‌ی اختصاص کار جدید به پردازنده، در صورتی که اجرای کار قبلی هنوز کامل نشده باشد، قبضه رخ داده و ذخیره می‌شود.	قبضه	<code>preemption</code>

تجاوز یک کار یا مجموعه‌ای از کارها از بدترین زمان اجرای خوش‌بینانه آن‌ها بعد از هر تیک پردازنده، بررسی می‌شود. در صورت رخداد تجاوز زمانی، اطلاعات کارها به همراه زمان رخداد ذخیره می‌شود.	تجاوز از زمان اجرا	overrun
بعد از هر تیک پردازنده، بررسی می‌شود که کار پرداش شده توسط پردازنده کامل شده است یا خیر. در صورت تکمیل کار، اطلاعات و زمان کامل شدن آن ذخیره می‌شود.	کامل شدن اجرای یک کار	jobFinish
بعد از هر بار فراخوانی پردازنده در یک واحد زمانی، بررسی می‌شود که کدام یک از کارهای آماده، مهلت زمانی خود را از دست داده‌اند. اطلاعات این کارها و زمان وقوع رخداد ذخیره می‌شود.	تجاوز از مهلت زمانی یک وظیفه	deadlineMiss
هر لحظه‌ای که تابع <code>dispatch</code> از کلاس زمان‌بند فراخوانی شود و کاری را به پردازنده اختصاص دهد، این رخداد ذخیره می‌شود.	اختصاص کار جدید به پردازنده	dispatch
در صورتی که کاری از درجه‌ی بحرانی بالا، مهلت واقعی اجرایش را از دست بدهد، زمان‌بندی ناموفق اعلام شده و با یک خطاب به پایان می‌رسد. این نتایج ذخیره می‌شوند.	شکست شبیه‌سازی	failure
در صورتی که بازه‌ی زمانی شبیه‌سازی بدون خطا کامل شود، شبیه‌سازی با موفقیت به پایان می‌رسد. در صورت پایان موفقیت‌آمیز یا ناموفق شبیه‌سازی، زمان‌بندی اعمال شده بر روی مجموعه‌ی وظایف، نمایش داده می‌شود.	زمان‌بندی کامل مجموعه وظایف	schedule
در هر لحظه که با انتشار کار جدید یا حذف کارهای LO، صف کارهای آماده به روزرسانی شود، صف جدید به کاربر نمایش داده می‌شود.	صف کارهای آماده	readyQ
با فراخوانی تابع <code>process</code> از پردازنده، در هر تیک از پردازنده، کاری پرداش می‌شود یا پردازنده بی کار است. وضعیت پردازنده در هر تیک با استفاده از این رخداد نمایش داده می‌شود.	تیک پردازنده	clock

## ۴ - ۵ - ۱۰ ارتباط بین کلاس‌ها

حال که با کلاس‌های سامانه به طور کامل آشنا شده‌ایم، می‌توانیم روابط بین آن‌ها را جمع‌بندی کنیم. کلاس اصلی و مدیریتی این سامانه، کلاس `Simulator` است که برنامه با ایجاد یک نمونه از آن شروع به کار می‌کند. در هنگام ساخته شدن، `Simulator` یک نمونه `ReadyQueue` و `CPU` و چند نمونه `Task` را ایجاد می‌کند. سپس یک نمونه از `Schedular` را می‌سازد و `CPU` و `ReadyQueue` را در اختیارش قرار می‌دهد. در انتهای با انتقال یک نمونه از `Task` به `ExecTimeGenerator`، یک نمونه از این کلاس را ایجاد می‌کند. کلاس `Logger` نیز به طور مجزا در ابتدای برنامه ساخته شده و به صورت سراسری در اختیار سایر کلاس‌ها قرار می‌گیرد اما `Simulator` قبل از شروع برنامه، اطلاعات `Task`‌ها، نمونه‌ی `Schedular` و همینطور `ReadyQueue` را در اختیار `Logger` قرار می‌دهد. هر `Task` با فراخوانی `Simulator`، نمونه‌هایی از `Job` را با کمک `ExecTimeGenerator` ایجاد می‌کند و در اختیار `ReadyQueue` قرار می‌دهد. این روابط به طور خلاصه در شکل ۷ بیان شده‌است.



شکل ۷ - نمودار ارتباط بین کلاس‌های نرم‌افزار شبیه‌ساز زمان‌بندی

## ۴ - ۶ ساختار پوشش‌های نرم‌افزار

در جدول ۷ لیستی از فایل‌ها و پوشش‌های مهم پروژه توضیح داده شده است. لازم به ذکر است که علاوه بر موارد ذکر شده در این جدول، فایل‌های دیگری نیز وجود دارند اما تلاش شده تا در این جدول، تنها به بخش‌های کلیدی اشاره شود.

جدول ۷ - لیست فایل‌های پروژه

توضیحات	نام فایل یا پوشش
پوشش حاوی کلاس‌های اصلی نرم‌افزار شبیه‌سازی است.	src/classes/
پوششی حاوی کلاس تولید زمان اجرای حقیقی و فایل‌های تولیدشده توسط آن، است.	src/jobs_data/
پوششی حاوی نتایج هر بار اجرای نرم‌افزار به ازای هر چهارتایی (مجموعه وظایف) - (نحوه تولید زمان اجرای حقیقی) - (طول زمان شبیه‌سازی) - (سیاست زمان‌بندی) با فرمت xml است.	src/result/
پوشش حاوی نرم‌افزار تولید مجموعه‌ی وظایف و خروجی‌های آن با فرمت xml است.	src/tasks_data/
تمامی تنظیمات پیکربندی سامانه در این فایل قابل مشاهده و تغییر می‌باشد.	src/configurations.ts
مقادیر تمامی تنظیمات پیکربندی سامانه را بررسی می‌کند و در صورت وجود ورودی غیرمجاز، خطای می‌دهد و از شروع نرم‌افزار جلوگیری می‌کند.	src/validate.ts

فایل اصلی پروژه می‌باشد که تولید مجموعه وظایف و شبیه‌سازی زمان‌بندی با اجرای آن شروع به کار می‌کند.	src/app.ts
مجموعه‌ای از توابع رایج استفاده شده در پروژه در این پوشه جمع‌آوری شده است؛ برای مثال، تابع تولید عدد صحیح تصادفی.	src/utils.ts
لیستی از ویژگی‌های اصلی پروژه مانند نام و نسخه و همچنین کتابخانه‌های لازم برای اجرای پروژه در این فایل جمع‌آوری شده است. همچنین دستور شروع پروژه نیز با نام simulate در این فایل تعریف شده است.	package.json

## ۴ - ۷ پیکربندی و دستورات اجرای نرم‌افزار

برای اجرای این نرم‌افزار ابتدا نیاز است که بسته‌های موردنیاز نصب شوند. به این منظور، باید در پوشه‌ی اصلی پروژه که فایل package.json در آن قرار دارد، دستور **yarn install** را در ترمینال یا خط فرمان، اجرا کرد. در اینجا پوشه‌های **yarn.lock** و **node\_modules** در میان فایل‌های پروژه ایجاد می‌شوند که به ترتیب حاوی نیازمندی‌های نصب شده و پیکربندی نصب آن‌ها می‌باشند.

پس از آن می‌توان نرم‌افزار پروژه را با وارد کردن دستور **yarn simulate** در همان خط فرمان، اجرا کرد. اجرای این دستور، فایل‌های پروژه را که با زبان تایپ‌اسکریپت پیاده‌سازی شده‌اند، پس از بررسی و کنترل نوع متغیرها و تولید خطا در صورت وجود ناهماهنگی، به جاوا اسکریپت ترجمه می‌کند و در پوشه‌ی **dist** می‌ریزد، سپس این فایل‌های جاوا اسکریپتی را اجرا می‌کند. نقطه‌ی شروع برنامه، فایل **app.ts** در پوشه‌ی **src** است.

قبل از شروع شبیه‌سازی می‌توان پیکربندی برنامه را از طریق فایل **configuration.ts** در سطوح شبیه‌سازی، زمان‌بندی، مجموعه وظایف و چاپ خروجی‌ها تغییر داد. توضیحات هر سطح به شرح زیر است:

- **شبیه‌سازی:** طول زمان اجرای شبیه‌سازی، حالت عملیاتی کلاس تولید زمان اجرای حقیقی و احتمال وقوع تجاوز از زمان اجرا را تنظیم می‌کند. اگر طول زمان اجرا باشد، زمان‌بندی اجرا نمی‌شود و نرم‌افزار فقط یک مجموعه وظایف تولید می‌کند.
- **زمان‌بندی:** مقدار اولیه‌ی تنظیمات سراسری زمان‌بندی را که در بخش ۴ - ۵ - ۱ - ۲ توضیح داده شد، تعیین می‌کند.
- **مجموعه وظایف:** ورودی‌های نرم‌افزار تولید مجموعه وظایف را مشخص می‌کند.
- **چاپ خروجی:** رخدادهای چاپ شده در خط فرمان را خاموش و روشن می‌کند.

---

---

با شروع اجرای برنامه، تمامی رخدادهای شبیه‌سازی در خط فرمان چاپ می‌شود. پس از پایان زمان شبیه‌سازی، زمان‌بندی انجام‌شده و داده‌های آماری در پوشه‌ی result ذخیره می‌شود. نتایج هر بار اجرای نرم‌افزار به ازای هر چهارتایی (شناسه مجموعه وظایف)–(نحوه تولید زمان‌های اجرای حقیقی)–(طول زمان شبیه‌سازی)–(سیاست زمان‌بندی) با فرمت xml در این پوشه ذخیره می‌شود.

به ازای تغییر هر کدام از ویژگی‌های مجموعه وظایف، یک مجموعه وظایف جدید با شناسه‌ی جدید تولید می‌شود که نتیجتاً منجر به تولید فایل‌های خروجی جدید می‌شود. تغییر در احتمال رخداد تجاوز از زمان اجرا، حالت عملیاتی تولید زمان‌های اجرای حقیقی، طول زمان شبیه‌سازی و سیاست زمان‌بندی که در بخش تنظیمات زمان‌بندی قرار دارد، منجر به تولید فایل‌های خروجی جدید بر همان مجموعه وظایف پیشین می‌شود.

## ۵ - ارزیابی پروژه

در این فصل به ارزیابی چارچوب نرمافزاری پیاده‌سازی شده برای زمان‌بندی سامانه‌های بحرانی مختلط می‌پردازیم و عملکرد آن را در شرایط مختلف بررسی می‌کنیم.

## ۵ - ۱ دریافت ورودی

همانطور که در فصل ۴ توضیح داده شد، تمامی ورودی‌های این نرمافزار پیش از اجرا در فایل configuration.ts دریافت می‌شود. از آنجایی که برخی از ورودی‌ها شروطی خاص دارند، تعریف و بررسی نوع آن‌ها به تنها‌ی برای جلوگیری از بروز خطا کافی نیست. برای مثال، قید شده است که تعداد وظایف باید مقداری عددی داشته باشد اما ۰ یا اعداد منفی نیز از نوع عدد هستند. از این‌رو، در فایل validate.ts از پوششی src، مقدار هر کدام از ورودی‌های پیکربندی نرمافزار بر اساس قوانین حاکم بر پیاده‌سازی، صحت‌سنگی می‌شود. در شکل ۸ - نشان داده شده که تلاش برای اجرای نرمافزار با یک پیکربندی اشتباه، دچار خطا می‌شود. اولین نقطه‌ای که صحت‌سنگ ورودی، خطا را در آن تشخیص داده است، تعداد وظایف است و در پیام خطا نیز به عبارت tasks.n اشاره می‌کند.

The screenshot shows a code editor with two tabs open. The left tab is 'configuration.ts' containing the following code:

```

11 const configurations : Configurations= {
12   duration: 10.5, // duration for which the simulation would run
13   overrunProbabilityPercentage: 50, // probability of actual C to be grater than C(L)
14   scheduling: {
15     exactOverrunTime: 1.5, // some integer time greater than 0, when this value is greater than 0, the overrunPossibility is ignored
16     overrunWatchingMechanism: "per_clock", // deprecated -> ignore
17     traditional: false, // traditional EDF instead of EDF-VO
18     workDonePerClock: 0.1, // indicating amount of work done in each clock -> main purpose: customizing the simulation for floating point values
19     frequency: 20, // f clock per time unit => (f * wpc) operation done in time unit = CPU Speed
20     initialSystemLevel: 10,
21   },
22   tasks: {
23     n: 0, // number of tasks
24     u: 1, // total system utilization
25     CF: 1.5, // criticality factor, > 1
26     CP: 2, // criticality proportion, < 1
27   }
28 };

```

The right tab is a terminal window titled 'Local (2)' showing the following command and its output:

```

Terminal: Local (2) + ▾
For more details, please visit https://support.apple.com/kb/HT208050.
maryam@final-project maryam$ yarn simulate
yarn run v1.22.19
$ tsc && node ./dist/app.js
/Users/maryam/Projects/final-project/dist/validate.js:10
    throw new Error(e);
    ^

```

An error message is displayed in the terminal:

```

Error: Some of the configuration values are illegal: tasks.n. Check your inputs in configurations.ts and try again!
at error (/Users/maryam/Projects/final-project/dist/validate.js:10:11)
at checkTasks (/Users/maryam/Projects/final-project/dist/validate.js:15:9)
at validate (/Users/maryam/Projects/final-project/dist/validate.js:47:5)
at Object.<anonymous> (/Users/maryam/Projects/final-project/dist/app.js:13:24)

```

شکل ۸ - نتایج کنترل ورودی‌های نرمافزار

## ۵ - ۲ تولید مجموعه وظایف

همانطور که در بخش ۴ - ۴ اشاره شد، نرمافزار می‌تواند با دریافت تعداد، بهره‌وری کل، دامنه‌ی تناوب، ضریب بحرانیت و احتمال بحرانیت، یک مجموعه وظایف بحرانی مختلط را تولید کند. به این منظور فایل configurations.ts را به صورت شکل ۹ تنظیم کرده و برنامه را اجرا می‌کنیم. نرمافزار بعد از ۱ دور تلاش،

مجموعه وظایفی با بهرهوری کلی ۰.۸۳۳ در حالت بحرانی پایین و بهرهوری کلی ۱.۶ در حالت بحرانی بالا تولید می‌کند. از آنجایی که نرمافزار تولید مجموعه وظایف، امکان‌پذیری هر وظیفه را بررسی و اعداد را با دقت ۰.۱ گرد می‌کند، میزان بهرهوری کل مجموعه‌ی تولیدشده کمی از بهرهوری کل درخواست‌شده کمتر خواهد بود.

مطابق ورودی‌ها، انتظار داریم یک مجموعه وظایف شامل ۴ وظیفه، با ۲ وظیفه از درجه‌ی بحرانی بالا و ۲ وظیفه از درجه‌ی بحرانی پایین، که بهرهوری کل آن نزدیک به ۱ باشد، تولید شود. دوره تناوب این وظایف نیز باید در بازه‌ی ۵ تا ۲۰ پراکنده شده باشد و بدترین زمان اجرای بدینانه هر وظیفه، در صورت امکان، ۲ برابر بدترین زمان اجرای خوشبینانه آن وظیفه باشد. همانطور که در شکل ۱۰ نشان داده شده، مجموعه وظایف تولید شده با شناسه‌ی [5-20]-4-1-2-0.5-[5-20]، دارای این مشخصات است.

```

> result
> tasks_data
> > out
> generator.ts
> samples.ts
> app.ts
configurations.ts
types.ts
utils.ts
validate.ts
ignore
package.json
tsconfig.json
yarn.lock
Terminal: Local | tasks-generation | + ▾
maryam:final-project maryam$ yarn simulate
yarn run v1.22.19
$ tsc && node ./dist/app.js
unsuccessful read from ./src/tasks_data/out/4-1-2-0.5-[5-20].xml: Error: ENOENT: no such file or directory, open './src/tasks_data/out/4-1-2-0.5-[5-20].xml'
DEBUG0: Found a feasible task set after 1 tries with actual utilization of: 1.60020202020202 | 0.83343434343434 instead of: 1
XML file written to ./src/tasks_data/out/4-1-2-0.5-[5-20].xml
task set generated to ./src/tasks_data/out/4-1-2-0.5-[5-20].xml
✖ Done in 2.65s.
maryam:final-project maryam$ 

```

شکل ۹ – پیکربندی و اجرای نرمافزار تولید مجموعه وظایف

```

<root>
  <tasks>
    <utilization>
      <L>0.5333333333333333</L>
      <H>1</H>
    </utilization>
    <period>5</period>
    <c>
      <L>0>4.8</L>
      <H>5</H>
    </c>
    <level>H1</level>
    <id>1</id>
  </tasks>
  <tasks>
    <utilization>
      <L>0>.09</L>
      <H>0.18</H>
    </utilization>
    <period>10</period>
    <c>
      <L>0>.9</L>
      <H>1.8</H>
    </c>
    <level>H2</level>
    <id>2</id>
  </tasks>
</root>

```

شکل ۱۰ – مجموعه وظایف تولیدشده توسط نرمافزار

## ۵ - ۳ شبیه‌سازی زمان‌بندی

در این بخش، مجموعه وظایف تولیدشده در بخش ۵ - ۲ را در حالت‌های مختلف زمان‌بندی می‌کنیم.

### ۵ - ۳ - ۱ زمان‌بندی سنتی تک‌بهرانیتی روی پردازنده‌ی تک‌هسته‌ای با سرعت واحد

در اینجا پیکربندی زمان‌بندی را به گونه‌ای تنظیم می‌کنیم که مجموعه وظایف بخش ۵ - ۲ را با الگوریتم EDF و به صورت تک‌بهرانیتی زمان‌بندی کند؛ به عبارتی، انتظار داریم یک شبیه‌سازی عادی از الگوریتم EDF را مشاهده کنیم. به این منظور، پیکربندی زمان‌بندی را برای ۹۹۰ واحد زمانی، زمان‌های اجرای واقعی ثابت و برابر با بدترین زمان اجرای خوش‌بینانه، تنظیم و برنامه را مطابق شکل ۱۱ اجرا می‌کنیم. فایل خروجی مطابق شکل ۱۲ و زمان‌بندی مطابق شکل ۱۳ است و نشان می‌دهد که طبق انتظار، در اجرای این مجموعه وظایف به صورت تک بهرانیتی و با زمان‌های اجرای ثابتی که برابر با بدترین زمان اجرای خوش‌بینانه هستند، تمام کارهای منتشر شده در مهلت زمانی خود به پایان می‌رسند. به این نکته توجه کنید که در آزمون زمان‌بندی این مجموعه وظایف امکان‌پذیر نیست، اما از آنجایی که زمان‌های اجرا در سطح بدترین زمان اجرای خوش‌بینانه تنظیم شده‌اند، شبیه‌سازی با موفقیت اجرا می‌شود.

```
maryam:final-project maryam$ yarn simulate
yarn run v1.22.19
$ tsc && node ./dist/app.js
task set loaded from ./src/tasks_data/out/4-1-2-0.5-[5-20].xml
{
  speed: 1,
  U11: 0.2101010101010101,
  U12: 0.4202020202020202,
  U21: 0.6233333333333333,
  U22: 1.18,
  u: -3.462962962962964,
  vdf: 0.7891304347826087,
  taskSetCheck: true,
  necessaryCheck: { result: false, method: '(U22 + U11) <= 1' },
  sufficientCheck: { result: true, method: '(U21 + U11) <= 1' }
}
~~~~~
This task set is not Schedulable
Strategy: traditional-edf
~~~~~
XML file written to ./src/result/(4-1-2-0.5-[5-20])-(L0)-(990)-(edf-traditional).xml
✖ Done in 2.25s.
```

شکل ۱۱ - اجرای زمان‌بندی سنتی تک‌بهرانیتی در پردازنده با سرعت واحد

ct	src	result	(4-1-2-0.5-[5-20])-(LO)-(990)-(edf-traditional)...
86	<schedule>	Analyzing...	<t-942>T1 (J94)</t-942>
87	<t-0>T1 (J0)</t-0>	727	<t-942.9>IDLE</t-942.9>
88	<t-4.8>T3 (J0)</t-4.8>	728	<t-945>T1 (J105)</t-945>
89	<t-5.3>T2 (J0)</t-5.3>	729	<t-946>T1 (J105)</t-946>
90	<t-6.2>T4 (J0)</t-6.2>	730	<t-949.8>T3 (J105)</t-949.8>
91	<t-7.9>IDLE</t-7.9>	731	<t-950>T3 (J105)</t-950>
92	<t-9>T1 (J1)</t-9>	732	<t-950.3>T4 (J86)</t-950.3>
93	<t-10>T1 (J1)</t-10>	733	<t-952>T2 (J95)</t-952>
94	<t-11>T1 (J1)</t-11>	734	<t-952.9>IDLE</t-952.9>
95	<t-13.8>T3 (J1)</t-13.8>	735	<t-954>T1 (J106)</t-954>
96	<t-14.3>T2 (J1)</t-14.3>	736	<t-957>T1 (J106)</t-957>
97	<t-15.2>T4 (J1)</t-15.2>	737	<t-958.8>T3 (J106)</t-958.8>
98	<t-16.9>IDLE</t-16.9>	738	<t-959.3>T4 (J87)</t-959.3>
99	<t-18>T1 (J2)</t-18>	739	<t-960>T4 (J87)</t-960>
100	<t-20>T1 (J2)</t-20>	740	<t-961>T2 (J96)</t-961>
101	<t-22>T1 (J2)</t-22>	741	<t-961.9>IDLE</t-961.9>
102	<t-22.8>T3 (J2)</t-22.8>	742	<t-963>T1 (J107)</t-963>
103	<t-23.3>T2 (J2)</t-23.3>	743	<t-967.8>T3 (J107)</t-967.8>
104	<t-24.2>T4 (J2)</t-24.2>	744	<t-968>T3 (J107)</t-968>
105	<t-25.9>IDLE</t-25.9>	745	<t-968.3>T4 (J88)</t-968.3>
106	<t-27>T1 (J3)</t-27>	746	<t-970>T2 (J97)</t-970>
107	<t-30>T1 (J3)</t-30>	747	<t-970.9>IDLE</t-970.9>
108	<t-31.8>T3 (J3)</t-31.8>	748	<t-972>T1 (J108)</t-972>
109	<t-32.3>T2 (J3)</t-32.3>	749	<t-976.8>T3 (J108)</t-976.8>
110	<t-33>T2 (J3)</t-33>	750	<t-977.3>IDLE</t-977.3>
111	<t-33.2>T4 (J3)</t-33.2>	751	<t-979>T4 (J89)</t-979>
112	<t-34.9>IDLE</t-34.9>	752	<t-980>T2 (J98)</t-980>
113	<t-36>T1 (J4)</t-36>	753	<t-980.9>T4 (J89)</t-980.9>
114	<t-40>T1 (J4)</t-40>	754	<t-981>T1 (J109)</t-981>
115	<t-40.8>T3 (J4)</t-40.8>	755	<t-985.8>T4 (J89)</t-985.8>
116	<t-41.3>T2 (J4)</t-41.3>	756	<t-986.4>T3 (J109)</t-986.4>
		757	<t-986.9>IDLE</t-986.9>
		758	

شکل ۱۲ – زمانبندی تولید شده برای حالت سنتی بر روی پردازنده با سرعت واحد

```

final-project > src > result > (4-1-2-0.5-[5-20])-(LO)-(990)-(edf-traditional)...
Project Analyzing...
Commit
Pull Requests
Structure
29      <HIbyHI>0</HIbyHI>
30      </preemptions>
31      <deadlineMisses>
32          <HI>0</HI>
33          <LO>0</LO>
34      </deadlineMisses>
35      <totalJobs>
36          <LO>200</LO>
37          <HI>209</HI>
38          <ignored>0</ignored>
39      </totalJobs>
40      <finishedJobs>
41          <HI>209</HI>
42          <LO>200</LO>
43      </finishedJobs>
44      <utilization>
45          <expected>
46              <U11>0.2101010101010101</U11>
47              <U21>0.6233333333333333</U21>
48              <U22>1.18</U22>
49          </expected>
50          <actual>
51              <U11>0.21010101010101007</U11>
52              <U12>0</U12>
53              <U21>0.6233333333333341</U21>
54              <U22>0</U22>
55          </actual>

```

شکل ۱۳ - نتایج آماری اجرای زمان‌بندی سنتی بر پردازنه با سرعت واحد

حال، کافی است شبیه‌سازی را کمی به واقعیت نزدیک‌تر کنیم و زمان اجرای واقعی هر وظیفه را مطابق با سطح بحرانیت همان وظیفه تعیین کنیم. به این ترتیب در پیکربندی شبیه‌سازی، مقدار ExectimeGeneratorMode را، که مشخص‌کنندهی حالت عملیاتی کلاس تولید زمان اجرای حقیقی است، به level تغییر می‌دهیم و شبیه‌سازی را دوباره اجرا می‌کنیم. شکل ۱۴ نتایج این شبیه‌سازی را نمایش می‌دهد که با خطا مواجه شده است. کار اول از وظیفه‌ی ۳، در لحظه‌ی ۹، مهلت اجرای خود را از دست می‌دهد.

```

const configurations : Configurations= {
  simulation: {
    duration: 990, // duration for which the simulation would run
    overrunProbabilityPercentage: 0, // probability of actual C to be grater than C(L0)
    ExecTimeGeneratorMode: "Level", Alikarami, Today * fix: multi-speed analysis
  },
  scheduling: {
    exactOverrunTime: 0, // some integer time greater than 0, when this value is greater than
    configurations > simulation > ExecTimeGeneratorMode
  }
}

task set loaded from ./src/tasks_data/out/4-1-2-0.5-[5-20].xml
{
  speed: 1,
  U11: 0.2101010101010101,
  U12: 0.4202020202020202,
  U21: 0.6233333333333333,
  U22: 1.18,
  u: -3.462962962962964,
  vdf: 0.7891304347826087,
  taskSetCheck: true,
  necessaryCheck: { result: false, method: '(U22 + U11) <= 1' },
  sufficientCheck: { result: true, method: '(U21 + U11) <= 1' }
}

This task set is not Schedulable
Strategy: traditional-edf
-----> 3-8 of level L0 (passing C(L0): 0.5 by -0.5 units overrun | d: 9) ] missed deadlines at: 9 System level: L0
XML file written to ./src/result/(4-1-2-0.5-[5-20])-level-(990)-(edf-traditional).xml
/Users/maryam/Projects/final-project/dist/classes/Simulator.js:71
    throw new Error("system failed!");

```

شکل ۱۴ – اجرای ناموفق زمان‌بندی سنتی بر مجموعه وظایف با بدترین زمان اجرای مناسب به درجهی بحرانیت

### ۵ - ۳ - ۲ - زمان‌بندی بحرانی مختلط با الگوریتم EDF-VD و مهلت‌های واقعی

در ادامه، تنظیمات زمان‌بندی مجموعه وظایف [5-20]-4-1-2-0.5 را با خاموش کردن حالت traditional برای زمان‌بندی بحرانی مختلط با الگوریتم EDF-VD تغییر می‌دهیم. نتیجه‌ی این اجرا در شکل ۱۵ آورده شده است. همانطور که مشاهده می‌کنید، این مجموعه وظایف حتی با تغییر حالت عملیاتی سامانه به درجهی بحرانی بالا هم قابل زمان‌بندی نیست و بررسی‌های اولیه‌ی انجام شده به روی مجموعه وظایف نیز این را نشان می‌دهند.

```

Project: final-project - ~/Projects/final-project master
Terminal: Local + v

exactOverrunTime: 0, // some integer time greater than 0, when this value is greater than
overrunWatchingMechanism: "per_clock", // deprecated -> ignore
traditional: false, // traditional EDF instead of EDF-VD You, 2 minutes ago * Uncommitted
workDonePerClock: 0.1, // indicating amount of work done in each clock -> main purpose: cu
frequency: 10. // f clock per time unit => (f * woc) operation_done in time_unit = CPU Spe
configurations > scheduling > traditional

{
  speed: 1,
  U11: 0.2101010101010101,
  U12: 0.4202020202020202,
  U21: 0.6233333333333333,
  U22: 1.18,
  u: -3.462962962962964,
  vdf: 0.7891304347826087,
  taskSetCheck: true,
  necessaryCheck: { result: false, method: '(U11 + U21 <= 1) && (U22 <= 1)' },
  sufficientCheck: {
    result: true,
    method: '(U11 + U22 <= 1): false OR (U11 + u <= 1): true'
  }
}

This task set is not MC-Schedulable
Strategy: edf-vd
-----> A jobs [ '1-0 (passing C(L0): 4.8 by 0 units overrun | d: 9)' ] overran at: 4.8
---> X ~ MODE CHANGE ~ X
---> A jobs [
  '2-0 of level HI (passing C(L0): 0.9 by 0.0999999999999998 units overrun | d: 10)'
] missed deadlines at: 10 System level: HI
XXXXXXXXXXXXXXXXXXXXXX
XML file written to ./src/result/(4-1-2-0.5-[5-20])-(level)-(990)-(edf-vd).xml
/Users/maryam/Projects/final-project/dist/classes/Simulator.js:71
    throw new Error("system failed!");

```

شکل ۱۵ – زمان‌بندی بحرانی مختلط با EDF-VD و مهلت‌های واقعی بر روی پردازنده با سرعت واحد

از آنجایی که ضریب افزایش سرعت EDF-VD حدود ۱.۷ می‌باشد، در مرحله‌ی بعد با افزایش فرکانس پردازنده از ۱۰ به ۱۷، سرعت پردازنده را از ۱ به ۱.۷ افزایش می‌دهیم و زمان‌بندی را دوباره اجرا می‌کنیم. انتظار داریم که الگوریتم در این حالت، قادر به زمان‌بندی مجموعه وظایف باشد. پیکربندی و نتایج این اجرا در شکل ۱۶ نشان داده شده است. بر اساس بررسی‌های اولیه، زمان‌بندی با الگوریتم EDF و مهلت‌های واقعی وظایف، امکان‌پذیر است. همانطور که مشاهده می‌کنید، سامانه در لحظه‌ی ۲.۸ وارد حالت بحرانی اجرا می‌شود اما زمان‌بندی با موفقیت به اتمام می‌رسد. با توجه به بزرگی زمان شبیه‌سازی، برنامه‌ی زمانی تولید شده، بسیار طولانی است و در این بخش از نمایش آن صرف نظر می‌شود.

```
Project: [Job.ts, ReadyQueue.ts, Scheduler.ts, Task.ts, utilization.ts, Logger.ts, Simulator.ts, System.ts]
Terminal: Local
$ tsc && node ./dist/app.js
task set loaded from ./src/tasks_data/out/4-1-2-0.5-[5-20].xml
{
  speed: 1.7000000000000002,
  U11: 0.1235888294711824,
  U12: 0.2471776589423648,
  U21: 0.3666666666666666,
  U22: 0.6941176470588234,
  u: 1.198717948717948,
  vdf: 0.41837288135593215,
  taskSetCheck: true,
  necessaryCheck: { result: true, method: '(U11 + U21 <= 1) && (U22 <= 1)' },
  sufficientCheck: {
    result: true,
    method: '(U11 + U22 <= 1): true OR (U11 + u <= 1): false'
  }
}

=====
This task set is MC-Schedulable
Strategy: edf
=====

--> ✅ jobs [ '1-0 (passing C(LO): 4.8 by 0 units overrun | d: 9)' ] overran at: 2.82353
--> ✖ ~ MODE CHANGE ~ ✖
XML file written to ./src/result/(4-1-2-0.5-[5-20])-level-(990)-(edf).xml
✖ Done in 2.49s.
maryam:final-project maryam$
```

شکل ۱۶ - زمان بندی بحرانی مختلط با EDF-VD و مهلت های واقعی پر روی پردازنده با سرعت ۱.۷

در مرحله‌ی قبل، زمان‌های واقعی اجرای هر وظیفه برابر با بدترین زمان اجرا در سطح اطمینان همان وظیفه قرار داده شده بود؛ در این مرحله، تنظیمات شبیه‌سازی را به شکلی تغییر می‌دهیم که زمان‌های اجرای حقیقی به صورت تصادفی و با احتمال تجاوز از بدترین زمان اجرای ۵ درصدی تولید شوند. انتظار داریم که در این مرحله نیز، سامانه وارد حالت بحرانی شود اما این اتفاق در لحظه‌ای بعد از ۲.۸ اتفاق بیافتد. پیکربندی و نتایج این اجرا در شکل ۱۷ آورده شده است و همانطور که مشاهده می‌شود، تغییر حالت سامانه در لحظه‌ی ۵.۰۶۰ اتفاق افتاده است. از آن‌جایی که بر خلاف مراحل قبلی، زمان‌های اجرا به صورت تصادفی تولید شدند، مشاهده می‌شود که این زمان‌ها در یک فایل xml در مسیر jobs\_data/out ذخیره شده‌اند. شکل ۱۸، بخشی از محتوای این فایل را نمایش می‌دهد.

```

$ tsc && node ./dist/app.js
task set loaded from ./src/tasks_data/out/4-1-2-0.5-[5-20].xml
unsuccessful read from ./src/jobs_data/out/4-1-2-0.5-[5-20](5%).xml: Error: ENOENT: no such file or directory, open './src/jobs_data/out/4-1-2-0.5-[5-20](5%).xml'
{
  speed: 1.7000000000000002,
  U11: 0.1235888294711824,
  U12: 0.2471776589423648,
  U21: 0.3666666666666666,
  U22: 0.6941176470588234,
  u: 1.198717948717948,
  vdf: 0.41837288135593215,
  taskSetCheck: true,
  necessaryCheck: { result: true, method: '(U11 + U21 <= 1) && (U22 <= 1)' },
  sufficientCheck: {
    result: true,
    method: '(U11 + U22 <= 1); true OR (U11 + u <= 1); false'
  }
}

This task set is MC-Schedulable
Strategy: edf
=====
--> A: jobs [ '2-6 (passing C(LO): 0.9 by 0 units overrun | d: 70)' ] overran at: 60.52941
--> X ~ MODE CHANGE ~ X
XML file written to ./src/jobs_data/out/4-1-2-0.5-[5-20](5%).xml
XML file written to ./src/result/(4-1-2-0.5-[5-20])-(random5%)-(990)-(edf).xml
+ Done in 2.38s.

```

شکل ۱۷ - اجرای زمان‌بندی EDF-VD با مهلت واقعی بر مجموعه وظایفی با احتمال تجاوز از زمان اجرای ۵ درصدی

```

<root> fatal: no such path src/jobs_data/out/4-1-2-0.5-[5-20](5%).xml in bb967fead49a849ef
<overrunPossibility>5</overrunPossibility>
<duration>990</duration>
<jobs>
  <id>1-0</id>
  <time>2.4</time>
</jobs>
<jobs>
  <id>2-0</id>
  <time>0.6</time>
</jobs>
<jobs>
  <id>3-0</id>
  <time>0.4</time>
</jobs>
<jobs>
  <id>4-0</id>
  <time>1.5</time>
</jobs>
<jobs>
  <id>1-1</id>
  <time>3.9</time>
</jobs>
<jobs>
  <id>3-1</id>
  <time>0.5</time>
</jobs>

```

شکل ۱۸ - خروجی زمان‌بندی EDF-VD با مهلت واقعی بر مجموعه وظایفی با احتمال تجاوز از زمان اجرای ۵ درصدی

### ۳ - ۵ - ۳ زمانبندی بحرانی مختلط با الگوریتم EDF-VD و مهلت‌های مجازی

در این بخش، تلاش می‌کنیم مجموعه وظایف را با حالت دوم از الگوریتم EDF-VD یعنی با مهلت‌های مجازی، زمانبندی کنیم. یک راه حل، کاهش فرکانس و در نتیجه سرعت پردازنده است. در بخش ۳ - ۳ - ۲ نشان دادیم که زمانبندی با مهلت‌های مجازی انجام می‌شود؛ اگر رابطه‌ی  $1 \leq U_1(1) + U_2(2)$  برقرار نباشد و رابطه‌ی  $1 \leq U_1(1) + \left(\frac{U_2(1)}{1-U_2(2)}\right)$  برقرار باشد. از طرف دیگر، در بخش ۵ - ۳ - ۲ مشاهده کردیم که مجموعه وظایف  $U_1(1) + U_2(2)$  تنها در پردازنده‌هایی با سرعت ۱.۷ و بزرگ‌تر از آن، قابل زمانبندی است. حاصل جمع  $U_1(1) + U_2(2)$  برای این مجموعه وظایف، طبق نتایج شکل ۱۴، «۱.۳۹» است. بنابراین هیچ‌گاه نمی‌توانیم با کاهش سرعت پردازنده، مجموعه وظایف مذکور را با الگوریتم EDF-VD با مهلت‌های مجازی، زمانبندی کنیم؛ چرا که در فرکانس‌های بالای ۱.۷، مجموعه همواره با مهلت واقعی زمانبندی می‌شود و در فرکانس‌های زیر ۱.۷ نیز زمانبندی امکان‌پذیر نیست در صورتی که برای زمانبندی با مهلت مجازی، فرکانس پردازنده باید کمتر از ۱۳.۹ باشد. در اینجا باید مجموعه وظایف جدیدی تولید کنیم. به این منظور ضریب بحرانیت را از ۰.۲ به ۰.۴، احتمال بحرانیت را از ۰.۵ به ۰.۴۴ و بهره‌وری کل را از ۰.۸ به ۰.۸۰ تغییر می‌دهیم و مجموعه‌ی [۵-۲۰]-۰.۸-۴-۰.۴۴ را مطابق شکل ۱۹ تولید می‌کنیم. از آنجایی که تناوب‌های جدید، ۷، ۱۷، ۱۶ و ۱۱ هستند، شبیه‌سازی‌های بعدی را برای ۲۰۹۴۴ واحد زمانی اجرا می‌کنیم.

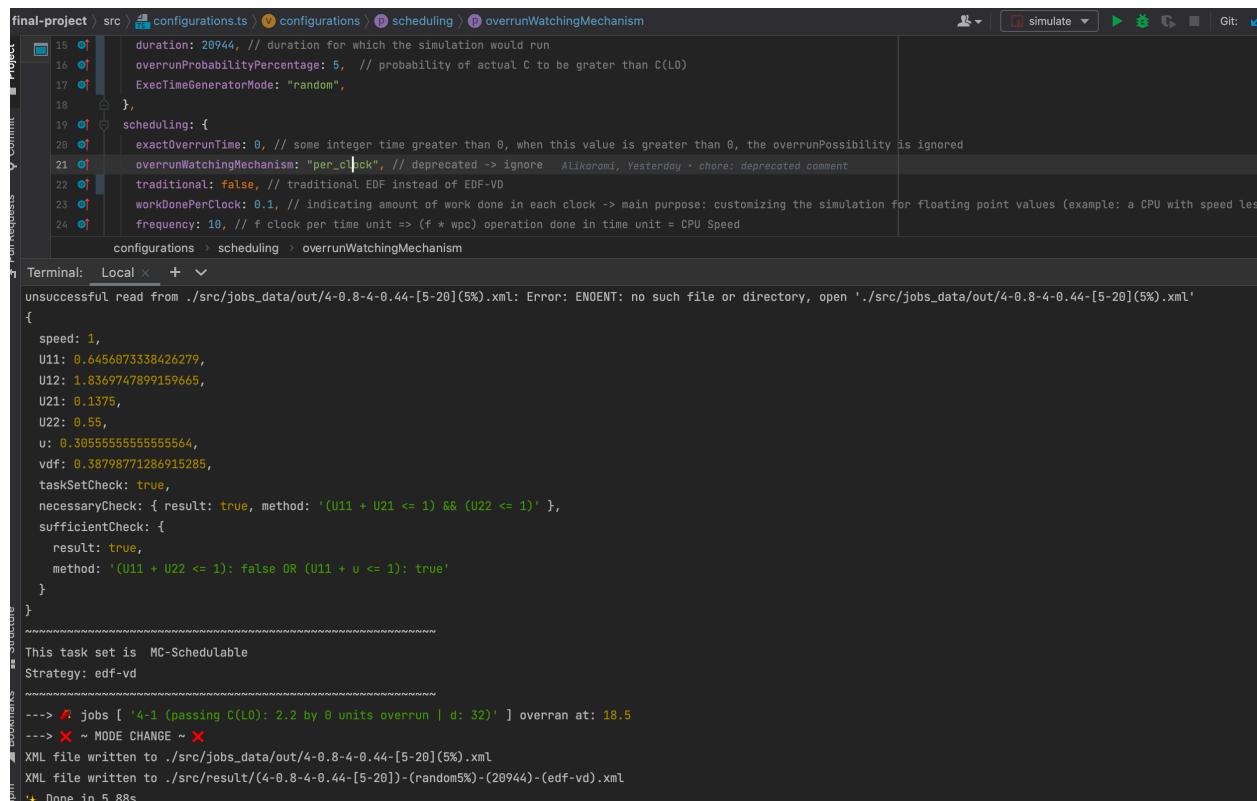
```

<root>
  <tasks>
    <utilization>
      <L0>0.18571428571428572</L0>
      <H1>0.7428571428571429</H1>
    </utilization>
    <period>7</period>
    <c>
      <L0>1.3</L0>
      <H1>5.2</H1>
    </c>
    <level>L0</level>
    <id>1</id>
  </tasks>
  <tasks>
    <utilization>
      <L0>0.43636363636363634</L0>
      <H1>1</H1>
    </utilization>
    <period>11</period>
    <c>
      <L0>4.8</L0>
      <H1>11</H1>
    </c>
    <level>L0</level>
    <id>2</id>
  </tasks>
  <tasks>
    <utilization>
      <L0>0.1375</L0>
      <H1>0.55</H1>
    </utilization>
    <period>16</period>
    <c>
      <L0>2.2</L0>
      <H1>8.8</H1>
    </c>
    <level>H1</level>
    <id>4</id>
  </tasks>
  <tasks>
    <utilization>
      <L0>0.023529411764705882</L0>
      <H1>0.09411764705882353</H1>
    </utilization>
    <period>17</period>
    <c>
      <L0>0.4</L0>
      <H1>1.6</H1>
    </c>
    <level>L0</level>
    <id>3</id>
  </tasks>
  <tasks>
    <utilization>
      <L0>0.1375</L0>
      <H1>0.55</H1>
    </utilization>
    <period>16</period>
    <c>
      <L0>2.2</L0>
      <H1>8.8</H1>
    </c>
    <level>H1</level>
    <id>4</id>
  </tasks>
</root>

```

شکل ۱۹ - مجموعه وظایف تولیدشده برای زمانبندی با مهلت مجازی

نتایج اجرای زمان‌بندی بر روی این مجموعه وظایف در شکل ۲۰ آمده است. همانطور که مشاهده می‌شود، مجموعه وظایف با الگوریتم EDF-VD و با استفاده از مهلت‌های مجازی زمان‌بندی شده است. تجاوز از زمان اجرا در لحظه‌ی ۱۸.۵ رخ می‌دهد و سامانه در این لحظه وارد حالت عملیاتی بحرانی می‌شود. در شکل ۲۱، تنظیمات خروجی را تغییر می‌دهیم تا قادر به مشاهده‌ی صفت کارهای آماده در حین اجرا باشیم. قابل مشاهده است که قبل از تغییر حالت عملیاتی سامانه، مهلت وظایف از درجه بحرانی بالا از مهلت واقعی آن‌ها کمتر و برابر با مهلت مجازی در نظر گرفته شده است و بعد از تغییر حالت سامانه، مهلت وظایف با مهلت واقعی آن‌ها برابر است و وظایف از درجه‌ی بحرانی پایین از صفت کارهای آماده حذف شده‌اند. بخشی از خروجی و داده‌های آماری این زمان‌بندی نیز در شکل ۲۲ آورده شده است. همانطور که مشاهده می‌شود، در اثر تغییر حالت سامانه، کار از درجه‌ی بحرانی پایین نادیده گرفته شده‌اند.



```

final-project > src > configurations.ts > configurations > scheduling > overrunWatchingMechanism
duration: 20944, // duration for which the simulation would run
overrunProbabilityPercentage: 5, // probability of actual C to be greater than C(L0)
ExecTimeGeneratorMode: "random",
},
{
  scheduling: {
    exactOverrunTime: 0, // some integer time greater than 0, when this value is greater than 0, the overrunPossibility is ignored
    overrunWatchingMechanism: "per_clock", // deprecated -> ignore Alikarami, Yesterday + chores: deprecated comment
    traditional: false, // traditional EDF instead of EDF-VD
    workDonePerClock: 0.1, // indicating amount of work done in each clock -> main purpose: customizing the simulation for floating point values (example: a CPU with speed less than 1)
    frequency: 10, // f clock per time unit => (f * wpc) operation done in time unit = CPU Speed
  }
}

configurations > scheduling > overrunWatchingMechanism

Terminal: Local < + >
unsuccessful read from ./src/jobs_data/out/4-0.8-4-0.44-[5-20](5%).xml: Error: ENOENT: no such file or directory, open './src/jobs_data/out/4-0.8-4-0.44-[5-20](5%).xml'
{
  speed: 1,
  U11: 0.6456073338426279,
  U12: 1.8369747899159665,
  U21: 0.1375,
  U22: 0.55,
  u: 0.3055555555555564,
  vdf: 0.38798771286915285,
  taskSetCheck: true,
  necessaryCheck: { result: true, method: '(U11 + U21 <= 1) && (U22 <= 1)' },
  sufficientCheck: {
    result: true,
    method: '(U11 + U22 <= 1): false OR (U11 + u <= 1): true'
  }
}
-----
This task set is MC-Schedulable
Strategy: edf-vd
-----
--> 4 jobs [ '4-1 (passing C(L0): 2.2 by 0 units overrun | d: 32)' ] overran at: 18.5
----> X ~ MODE CHANGE ~ X
XML file written to ./src/jobs_data/out/4-0.8-4-0.44-[5-20](5%).xml
XML file written to ./src/result/(4-0.8-4-0.44-[5-20])-(random5%)-(20944)-(edf-vd).xml
+ Done in 5.88s.

```

شکل ۲۰ – اجرای زمان‌بندی EDF-VD با مهلت مجازی بر روی ۴-۰.۸-۴-۰.۴۴-[۵-۲۰]

```
[{"id": 2-1, deadline: 22, actualDeadline: 22, level: L0, executed: 0, releasedAt: 11, actualC: 4.4, WCET(L0): 4.8, T: 11"}, {"id": 2-1, deadline: 22, actualDeadline: 22, level: L0, executed: 3, releasedAt: 11, actualC: 4.4, WCET(L0): 4.8, T: 11"}, {"id": 1-2, deadline: 21, actualDeadline: 21, level: L0, executed: 0, releasedAt: 14, actualC: 0.9, WCET(L0): 1.3, T: 7"}, {"id": 2-1, deadline: 22, actualDeadline: 22, level: L0, executed: 4.1, releasedAt: 11, actualC: 4.4, WCET(L0): 4.8, T: 11"}, {"id": 4-1, deadline: 22.2, actualDeadline: 32, level: HI, executed: 0, releasedAt: 16, actualC: 4.2, WCET(L0): 2.2, T: 16"}, {"id": 4-1, deadline: 22.2, actualDeadline: 32, level: HI, executed: 0.7, releasedAt: 16, actualC: 4.2, WCET(L0): 2.2, T: 16}, {"id": 3-1, deadline: 34, actualDeadline: 34, level: L0, executed: 0, releasedAt: 17, actualC: 0.3, WCET(L0): 0.4, T: 17}], [{"jobs": "4-1 (passing CLO): 2.2 by 0 units overrun | d: 32", "overrun": 18.5}], [{"reason": "MODE CHANGE"}], [{"job": "4-1, deadline: 32, actualDeadline: 32, level: HI, executed: 2.2, releasedAt: 16, actualC: 4.2, WCET(HI): 8.8, T: 16"}], [{"job": "4-2, deadline: 48, actualDeadline: 48, level: HI, executed: 0, releasedAt: 32, actualC: 1.8, WCET(HI): 8.8, T: 16"}], [{"job": "4-3, deadline: 64, actualDeadline: 64, level: HI, executed: 0, releasedAt: 48, actualC: 1.5, WCET(HI): 8.8, T: 16"}]
```

شکل ۲۱ – صف کارهای آماده در حین اجرای الگوریتم EDF-VD با مهلت مجازی بر روی [5-20]

```
<schedule>
  <t-0>T4 (J0)</t-0>
  <t-2>T1 (J0)</t-2>
  <t-2.8>T2 (J0)</t-2.8>
  <t-6.6>T3 (J0)</t-6.6>
  <t-6.9>IDLE</t-6.9>
  <t-7>T1 (J1)</t-7>
  <t-7.8>IDLE</t-7.8>
  <t-11>T2 (J1)</t-11>
  <t-14>T1 (J2)</t-14>
  <t-14.9>T2 (J1)</t-14.9>
  <t-16>T2 (J1)</t-16>
  <t-16.3>T4 (J1)</t-16.3>
  <t-17>T4 (J1)</t-17>
  <t-18.5>T4 (J1)</t-18.5>
  <t-20.5>IDLE</t-20.5>
  <t-32>T4 (J2)</t-32>
  <t-33.8>IDLE</t-33.8>
  <t-48>T4 (J3)</t-48>
  <t-49.5>IDLE</t-49.5>
  <t-64>T4 (J4)</t-64>
  <t-65.5>IDLE</t-65.5>
  <t-80>T4 (J5)</t-80>
  <t-81.8>IDLE</t-81.8>
  <t-96>T4 (J6)</t-96>
  <t-97.3>IDLE</t-97.3>
  <t-112>T4 (J7)</t-112>
  <t-113.1>IDLE</t-113.1>
  <t-128>T4 (J8)</t-128>
  <t-132.2>IDLE</t-132.2>
  <t-136>T4 (J9)</t-136>
```

<modeChange>  
     <time>18.5</time>  
     <reason>overrun</reason>  
 </modeChange>  
 <preemptions>  
     <L0byL0>1</L0byL0>  
     <L0byH1>0</L0byH1>  
     <H1byL0>0</H1byL0>  
     <H1byH1>0</H1byH1>  
 </preemptions>  
 <deadlineMisses>  
     <H1>0</H1>  
     <L0>0</L0>  
 </deadlineMisses>  
 <totalJobs>  
     <L0>7</L0>  
     <H1>1309</H1>  
     <ignored>6121</ignored>  
 </totalJobs>  
 <finishedJobs>  
     <H1>1309</H1>  
     <L0>6</L0>  
 </finishedJobs>  
 <utilization>  
     <expected>  
         <U11>0.6456073338426279</U11>  
         <U21>0.1375</U21>  
         <U22>0.55</U22>  
     </expected>  
 <actual>  
     <U11>0.0005252100846336136</U11>

شکل ۲۲ – داده‌های آماری اجرای EDF-VD با مهلت مجازی بر روی [5-20]

می‌توان با تنظیم احتمال وقوع تجاوز از مهلت اجرا به صفر درصد در کلاس تولید زمان اجرای واقعی و تولید زمان‌های اجرایی که همواره از بدترین زمان اجرای منتبس به درجهٔ بحرانیت هر وظیفه کوچک‌تر هستند، همین زمان‌بندی را مجدداً به گونه‌ای اجرا کرد که تغییر حالت عملیاتی اتفاق نیافتد. در شکل ۲۳ خروجی‌های این اجرا

نمایش داده شده است. واضح است که در این حالت، بهرهوری سامانه بسیار بالاتر است و بدون حتی یک مهلت از دست رفته، سامانه تمامی کارهای منتشر را با الگوریتم EDF-VD و مهلت‌های مجازی، زمان‌بندی می‌کند.

```

<statistics>
  <analysis>
    <speed>1</speed>
    <U11>0.6456073338426279</U11>
    <U12>1.8369747899159665</U12>
    <U21>0.1375</U21>
    <U22>0.55</U22>
    <U>0.30555555555555564</U>
    <vdf>0.38798771286915285</vdf>
    <taskSetCheck>true</taskSetCheck>
    <necessaryCheck>true</necessaryCheck>
    <sufficientCheck>true</sufficientCheck>
    <mechanism>edf-vd</mechanism>
  </analysis>
  <modeChange/>
  <preemptions>
    <L0byL0>697</L0byL0>
    <L0byH1>399</L0byH1>
    <HIbyL0>0</HIbyL0>
    <HIbyH1>0</HIbyH1>
  </preemptions>
  <deadlineMisses>
    <HI>0</HI>
    <L0>0</L0>
  </deadlineMisses>
  <totalJobs>
    <L0>6128</L0>
    <HI>1309</HI>
    <ignored>0</ignored>
  </totalJobs>
  <finishedJobs>
    <HI>1309</HI>
    <L0>6128</L0>
  </finishedJobs>
</utilization>
<responseTime>
  <T-1>
    <min>0.7</min>
    <max>3.9</max>
    <avg>1.1841243315507972</avg>
  </T-1>
  <T-2>
    <min>2.4</min>
    <max>8.1</max>
    <avg>4.65126050420167</avg>
  </T-2>
  <T-3>
    <min>0.2</min>
    <max>9.4</max>
    <avg>2.009983766233766</avg>
  </T-3>
  <T-4>
    <min>1.1</min>
    <max>2.7</max>
  </T-4>
</responseTime>

```

شکل ۲۳ – داده‌های آماری اجرای EDF-VD با مهلت مجازی بر روی [۵-۲۰] با احتمال صفر درصدی تجاوز از زمان اجرا

### ۴ - ۳ - ۵ ورود اجباری سامانه به حالت عملیاتی بحرانی

این نرم‌افزار به کاربر اجازه می‌دهد تا برای تولید زمان‌بندی‌های ایستا، لحظه‌ی دقیق وقوع تجاوز از زمان اجرا و ورود سامانه به حالت عملیاتی بحرانی را تعیین کند. در این لحظه، بدون توجه به اینکه تجاوز از زمان اجرا واقعاً رخ داده است یا خیر، سامانه وارد حالت عملیاتی بحرانی می‌شود و وظایف از درجه‌ی بحرانی پایین حذف می‌شوند. البته در صورتی که زمان‌های اجرای حقیقی با احتمال بالایی از بدترین زمان اجرای خود تجاوز کنند، ممکن است که تغییر حالت قبل از لحظه‌ی تعیین شده رخ بدهد؛ به همین دلیل، کاربر باید در تنظیم پیکربندی سامانه دقت کند. با پیکربندی مشابه شکل ۲۴، برنامه را اجرا می‌کنیم، روند اجرا در شکل ۲۵ و خروجی در شکل ۲۶ نشان داده شده است. طبق انتظار، سامانه در لحظه‌ی مشخص شده‌ی ۲، تغییر حالت می‌دهد و دلیل آن نیز ذکر شده است. همچنین از آنجایی که تمام سطوح چاپ خروجی فعال هستند، روند اجرای نرم‌افزار با ریزترین جزئیات اجرای هر تیک از پردازنده، قابل مشاهده است.

```

const configurations : Configurations= {
  simulation: {
    duration: 5, // duration for which the simulation runs
    overrunProbabilityPercentage: 0, // probability of an overrun
    execTimeGeneratorMode: "Level",
  },
  scheduling: {
    exactOverrunTime: 2, // some integer time greater than zero
    overrunWatchingMechanism: "per_clock", // depends on the clock
    traditional: false, // traditional EDF instead
    workDonePerClock: 0.1, // indicating amount of work done per clock
    frequency: 10, // f clock per time unit => (f * workDonePerClock) = utilization
    initialSystemLevel: L0,
  },
  tasks: {
    n: 4, // number of tasks
    u: 0.8, // total system utilization
    CF: 4, // criticality factor, > 1
    CP: 0.44, // criticality proportion, < 1
    minPeriod: 5,
    maxPeriod: 20,
  },
}
  
```

32     minPeriod: 5,  
33     maxPeriod: 20,  
34     },  
35     log: {  
36         enabled: true,  
37         setting: {  
38             time: true,  
39             utilization: true,  
40             arrival: true,     You, A minute ago + Uncommitted changes  
41             feasibilityTest: true,  
42             preemption: true,  
43             overrun: true,  
44             jobFinish: true,  
45             deadlineMiss: true,  
46             dispatch: true,  
47             failure: true,  
48             schedule: true,  
49             readyQ: true,  
50             clock: true,  
51         }  
52     }  
53 }

شکل ۲۴ – پیکربندی سامانه برای ورود اجباری به حالت عملیاتی بحرانی در لحظه ۲

```

Terminal: Local + v
=====
This task set is MC-Schedulable
Strategy: edf-vd
*****
* job: 1-0 arrived at 0
* job: 2-0 arrived at 0
* job: 3-0 arrived at 0
* job: 4-0 arrived at 0
[

  [[id: 1-0, deadline: 7, actualDeadline: 7, level: L0, executed: 0, releasedAt: 0, actualC: 1.3, WCET(L0): 1.3, T: 7]],
  [[id: 2-0, deadline: 11, actualDeadline: 11, level: L0, executed: 0, releasedAt: 0, actualC: 4.8, WCET(L0): 4.8, T: 11]],
  [[id: 3-0, deadline: 17, actualDeadline: 17, level: L0, executed: 0, releasedAt: 0, actualC: 0.4, WCET(L0): 0.4, T: 17]],
  [[id: 4-0, deadline: 6.2, actualDeadline: 16, level: HI, executed: 0, releasedAt: 0, actualC: 8.8, WCET(L0): 2.2, T: 16]]
]

■ job: 4-0 dispatched at 0
c0: running 4-0 from 0 to 0.1 (remaining: 8.7)
c1: running 4-0 from 0.1 to 0.2 (remaining: 8.6)
c2: running 4-0 from 0.2 to 0.3 (remaining: 8.5)
c3: running 4-0 from 0.3 to 0.4 (remaining: 8.4)
c4: running 4-0 from 0.4 to 0.5 (remaining: 8.3)
c5: running 4-0 from 0.5 to 0.6 (remaining: 8.2)
c6: running 4-0 from 0.6 to 0.7 (remaining: 8.1)
c7: running 4-0 from 0.7 to 0.8 (remaining: 8.0)
c8: running 4-0 from 0.8 to 0.9 (remaining: 7.9)
c9: running 4-0 from 0.9 to 1 (remaining: 7.8)
-----
***** 1 *****
c0: running 4-0 from 1 to 1.1 (remaining: 7.7)
c1: running 4-0 from 1.1 to 1.2 (remaining: 7.6)

=====
***** 2 *****
c2: running 4-0 from 1.2 to 1.3 (remaining: 7.5)
c3: running 4-0 from 1.3 to 1.4 (remaining: 7.4)
c4: running 4-0 from 1.4 to 1.5 (remaining: 7.3)
c5: running 4-0 from 1.5 to 1.6 (remaining: 7.2)
c6: running 4-0 from 1.6 to 1.7 (remaining: 7.1)
c7: running 4-0 from 1.7 to 1.8 (remaining: 7.0)
c8: running 4-0 from 1.8 to 1.9 (remaining: 6.9)
c9: running 4-0 from 1.9 to 2 (remaining: 6.8)
-----
***** 3 *****
c0: running 4-0 from 3 to 3.1 (remaining: 5.7)
c1: running 4-0 from 3.1 to 3.2 (remaining: 5.6)
c2: running 4-0 from 3.2 to 3.3 (remaining: 5.5)
c3: running 4-0 from 3.3 to 3.4 (remaining: 5.4)
c4: running 4-0 from 3.4 to 3.5 (remaining: 5.3)
c5: running 4-0 from 3.5 to 3.6 (remaining: 5.2)
c6: running 4-0 from 3.6 to 3.7 (remaining: 5.1)
c7: running 4-0 from 3.7 to 3.8 (remaining: 5.0)
c8: running 4-0 from 3.8 to 3.9 (remaining: 4.9)
c9: running 4-0 from 3.9 to 4 (remaining: 4.8)
-----
***** 4 *****
c0: running 4-0 from 4 to 4.1 (remaining: 4.7)
c1: running 4-0 from 4.1 to 4.2 (remaining: 4.6)
c2: running 4-0 from 4.2 to 4.3 (remaining: 4.5)
c3: running 4-0 from 4.3 to 4.4 (remaining: 4.4)
c4: running 4-0 from 4.4 to 4.5 (remaining: 4.3)
c5: running 4-0 from 4.5 to 4.6 (remaining: 4.2)
c6: running 4-0 from 4.6 to 4.7 (remaining: 4.1)
c7: running 4-0 from 4.7 to 4.8 (remaining: 4.0)
c8: running 4-0 from 4.8 to 4.9 (remaining: 3.9)
c9: running 4-0 from 4.9 to 5 (remaining: 3.8)
-----
-----SCHEDULE-----
0 <- T4 (J0)
XML file written to ./src/result/(4-0.8-4-0.44-[5-20])-{level}-{5}-{edf-vd}.xml
* Done in 2.69s.
maryam:final-project maryam$ 
  
```

شکل ۲۵ – اجرای شبیه‌سازی برای ورود اجباری سامانه به حالت عملیاتی بحرانی در لحظه ۲

```

<workDonePerClock>0.1</workDonePerClock>
<frequency>10</frequency>
<initialSystemLevel>L0</initialSystemLevel>
<config>
<result>success</result>
<statistics>
    <analysis>
        <speed>1</speed>
        <U11>0 .6456873338426279</U11>
        <U12>1 .8369747899159665</U12>
        <U21>0 .1375</U21>
        <U22>0 .55</U22>
        <u>0 .3055555555555564</u>
        <vdf>0 .38798771286915285</vdf>
        <taskSetCheck>true</taskSetCheck>
        <necessaryCheck>true</necessaryCheck>
        <sufficientCheck>true</sufficientCheck>
        <mechanism>edf-vd</mechanism>
    </analysis>
    <modeChange>
        <time>2</time>
        <reason>forced</reason>
    </modeChange>
    <preemptions>
        <L0byL0>0</L0byL0>
        <L0byH1>0</L0byH1>
        <H1byL0>0</H1byL0>
        <H1byH1>0</H1byH1>
    </preemptions>

```

root > statistics > analysis

```

</utilization>
<responseTime>
    <T-1>
        <min/>
        <max>Infinity</max>
        <avg>NaN</avg>
    </T-1>
    <T-2>
        <min/>
        <max>Infinity</max>
        <avg>NaN</avg>
    </T-2>
    <T-3>
        <min/>
        <max>Infinity</max>
        <avg>NaN</avg>
    </T-3>
    <T-4>
        <min/>
        <max>Infinity</max>
        <avg>NaN</avg>
    </T-4>
</responseTime>
</statistics>
<schedule>
    <t-0>T4 (J0)</t-0>
</schedule>
</root>

```

root > statistics > analysis

شکل ۲۶ – داده‌های آماری شبیه‌سازی ورود اجباری سامانه به حالت عملیاتی بحرانی در لحظه ۲

## ۴ - ۵ محدودیت‌های نرم‌افزار

همانطور که مشاهده شد، این نرم‌افزار قابلیت تولید و شبیه‌سازی زمان‌بندی سنتی و بحرانی مختلط یک مجموعه وظایف بحرانی مختلط بر روی پردازنده‌ای تک‌هسته با سرعت متغیر را داراست و با پیکربندی گسترده و محاسبات آماری، امکان مقایسه‌ی تاثیر مولفه‌های گوناگون بر زمان‌بندی سامانه را فراهم می‌کند. همچنین با ذخیره‌ی مقادیر تولیدشده‌ی تصادفی، می‌تواند در دفعات متعدد اجرای یک پیکربندی، نتایج ثابتی را تولید کند. از این رو می‌توان ادعا کرد که این نرم‌افزار در زمینه‌ی شبیه‌سازی زمان‌بندی، سامانه‌ای قدرتمند و انعطاف‌پذیر است. در عین حال محدودیت‌هایی نیز دارد که در زیر به آن اشاره می‌کنیم:

- پیاده‌سازی نرم‌افزار بر مبنای زمان‌بندی یک پردازنده‌ی تک‌هسته‌ای بنا شده است و امکان زمان‌بندی چند هسته‌ای را دارا نیست. به این منظور نیاز است تا سازوکاری برای اختصاص وظایف به هسته‌های گوناگون و همچنین اجرای موازی شبیه‌سازی بر روی آن هسته‌ها فراهم شود.
- این نرم‌افزار مجموعه وظایفی مستقل را مدل می‌کند که در آن تقدم و تاخر وظایف، منوط به رعایت نیازمندی‌های زمان‌بندی، تاثیری بر اجرا ندارد و هیچ وظیفه‌ای به دیگری وابسته نیست. برای شبیه‌سازی یک مجموعه وظایف که در آن، اجرای برخی از کارها به کارهای دیگر وابسته است، مجموعه وظایف باید

به صورت دیگری مدل شود و نرمافزار امکان پشتیبانی از مدل‌های تعریف وظایف وابسته مانند<sup>۱</sup> DAG و MC-DAG<sup>۲</sup> را داشته باشد. بررسی این گونه مجموعه وظایف از توضیح این پروژه خارج است.

- دقت اعداد ورودی این نرمافزار تا یک مرحله‌ی اعشار می‌باشد و از این نظر محدودیت‌هایی را در تعریف و تولید مجموعه وظایف ایجاد می‌کند؛ برای مثال، WCET‌های تولیدشده در فرآیند ایجاد مجموعه وظایف با دقت ۰.۱ گرد می‌شود. این باعث می‌شود که بهره‌وری کل به دست آمده از بهره‌وری کل خواسته شده کوچک‌تر باشد و از میزان دقت سامانه کاسته شود.

این نرمافزار به صورت همگام پیاده‌سازی شده و شبیه‌سازی در دو حلقه‌ی تودرتو انجام می‌شود اما برای شبیه‌سازی دقیق‌تر از رفتار سامانه‌های بی‌درنگ در دنیای واقعی، پیاده‌سازی رویداد-محور بهترین گزینه است. از آنجایی که تایپ‌اسکریپت زبان این نرمافزار است، بستر چنین گسترشی در پیاده‌سازی فراهم است.

این نرمافزار تنها از دو سطح بحرانی و دو حالت عملیاتی عادی و بحرانی پشتیبانی می‌کند در صورتی که سامانه‌های بحرانی مختلط می‌توانند با بیشتر از دو سطح، تعریف شوند. پیاده‌سازی این نرمافزار با این دو حالت بحرانی بسیار در هم‌تنیده است و آن‌ها را با دو مقدار رشته‌ای LO و HI تعریف کرده‌است. از این رو گسترش نرمافزار برای پشتیبانی از تعداد بیشتر سطوح، به بازسازی و بازنمایی بعضی از مفاهیم و متغیرها نیازمند است.

در این پیاده‌سازی، نرمافزار تنها به حالت بحرانی عملیاتی تغییر حالت می‌دهد و به حالت عملیاتی عادی باز نمی‌گردد؛ گرچه می‌توان چنین سازوکاری را به راحتی با تعریف مقادیر پیکربندی جدید به این نرمافزار اضافه کرد، اما در حال حاضر از این ویژگی پشتیبانی نمی‌کند.

در این نرمافزار، تمرکز بر پیاده‌سازی سیاست‌های الگوریتم زمان‌بندی EDF و EDF-VD بوده است. اگرچه نرمافزار از نظر معماری و تعریف اجزاء، بستری آمده برای افزودن سیاست‌های زمان‌بندی متنوع دارد، اما در حال حاضر از الگوریتم‌های بیشتری پشتیبانی نمی‌کند.

<sup>۱</sup> Direct Acyclic Graph

<sup>۲</sup> Mixed Criticality Direct Acyclic Grapah

## ٦ - جمع‌بندی، نتیجه‌گیری و پیشنهادات

## ۶ - ۱ جمع‌بندی و نتیجه‌گیری

سامانه‌های بحرانی مختلط نوعی از سامانه‌های بی‌درنگ هستند که ضمن اجرای وظایف از درجه‌ی بحرانی متفاوت بر یک بستر یکسان، اینمی سامانه و اطمینان از اجرای به موقع وظایف بحرانی را نیز تضمین می‌کنند. روش‌های بحرانی مختلط بر آن هستند که تعادلی میان دو جنبه‌ی کارایی و اینمی ایجاد کنند. از آنجایی که تضمین اجرای وظایف بحرانی-ایمن برای دریافت گواهینامه‌های استاندارد از یک سو و صدور اجازه‌ی اجرای وظایف از بحران پایین‌تر در کنار یا پیش از وظایف بحرانی-ایمن برای افزایش کارایی سامانه از سوی دیگر، با هم در تقابل هستند، پیاده‌سازی سامانه‌های بحرانی مختلط با رعایت شرط عدم تداخل وظایف از بحران‌های متفاوت، امری چالش برانگیز است. در این گزارش، سامانه‌های بحرانی مختلط را به عنوان نوعی از سامانه‌های بی‌درنگ معرفی کردیم و چالش‌های پیاده‌سازی و صنعتی‌سازی این سامانه‌ها را برشمردیم و نشان دادیم که ارائه‌ی تعریف واحدی از بحرانیت، یکی از اساسی‌ترین چالش‌های صنعتی‌سازی این سامانه‌ها است.

با تمرکز بر سامانه‌های دو-بحرانیتی، زمان‌بندی سامانه‌های بحرانی مختلط را بررسی کردیم و با ارائه‌ی مثال‌هایی نشان دادیم که الگوریتم‌های سنتی تک-بحرانیتی مانند EDF و DM برای زمان‌بندی این سامانه‌ها مناسب نیستند. الگوریتم‌هایی که تخصیص اولویت به وظایف را صرفا بر اساس درجه‌ی بحرانیت آن‌ها انجام می‌دهند و وظایف بحرانی را همواره قبل از وظایف عادی اجرا می‌کنند، گرچه در تضمین اینمی و عدم تداخل وظایف موفق هستند. اما قدرت زمان‌بندی کمی دارند و قادر به زمان‌بندی طیف وسیعی از مجموعه وظایف با بحرانیت مختلط نیستند. همچنین نشان دادیم که تعریف درجه‌ی بحرانیت برای سطح عملیات سامانه می‌تواند زمان‌بندی‌پذیری سامانه را افزایش دهد. صنعتی‌سازی این روش‌ها نیز در صورتی امکان‌پذیر است که تخصیص بحرانیت به وظایف به درستی انجام شده باشد و طراح سامانه از بدترین زمان‌های خوش‌بینانه‌ای که به وظایف اختصاص داده است، اطمینان بالایی داشته باشد چراکه در واقعیت انتظار نداریم که وظایف از این زمان تجاوز کنند و سامانه وارد حالت عملیاتی بحرانی شود.

در ادامه‌ی مسیر تکامل الگوریتم‌های زمان‌بندی بحرانی مختلط، الگوریتم EDF-VD را معرفی کردیم. با استفاده از چارچوب شبیه‌سازی زمان‌بندی خود، نشان دادیم که این الگوریتم عملکردی بهتر از EDF دارد و با ضریب افزایش سرعت ۱.۷ قادر به زمان‌بندی مجموعه وظایفی است که بر پردازنده‌ای با سرعت واحد، زمان‌بندی‌پذیر نیستند. همچنین نشان دادیم که پیکربندی‌های متنوع چارچوب شبیه‌سازی ما برای مقایسه‌ی قدرت زمان‌بندی الگوریتم‌های متفاوت بر یک مجموعه وظایف، کارا و مناسب است و می‌توانیم با این چارچوب، تاثیرگذاری مولفه‌های متفاوت سامانه مانند سرعت و فرکانس پردازنده را بر زمان‌بندی‌پذیری مجموعه وظایف بسنجیم. چارچوب زمان‌بندی ما از یک سو می‌تواند برای زمان‌بندی وظایف بحرانی مختلط به صورت ایستا یا آنلاین مورد استفاده

قرار گیرد و از سوی دیگر، با ارائه‌ی خروجی‌های آماری، بستری مناسب برای مقایسه‌ی الگوریتم‌های زمان‌بندی بحرانی مختلط و تاثیر عوامل محیطی بر آن‌ها باشد.

## ۶ - ۲ پیشنهادها

با وجود نتایج امیدوارکننده، پیشنهاداتی برای گسترش چارچوب نرم‌افزاری زمان‌بندی بحرانی مختلطی که در این پروژه ارائه شد و همچنین الگوریتم EDF-VD وجود دارد:

- آزمایش در شرایط واقعی: کاربرد الگوریتم EDF-VD در سناریوهای واقعی در صنایع مختلف می‌تواند بینش‌های عمیق‌تری در مورد کاربردها و محدودیت‌های عملی آن فراهم آورد. به این منظور به بررسی مجموعه وظایفی صنعتی نیاز است که داده‌های آن به آسانی قابل دستیابی نیست.
- تجزیه و تحلیل مقایسه‌ای: چارچوب شبیه‌سازی‌شده در این پروژه قادر است الگوریتم EDF و EDF-VD را به سادگی مورد مقایسه قرار دهد اما گسترش تنظیمات آن برای مقایسه‌ی الگوریتم EDF-VD با سایر الگوریتم‌های نوظهور، می‌تواند قدم مثبتی در راستای تحقیقات انجام‌شده بر زمان‌بندی سامانه‌های بحرانی مختلط باشد و چارچوب مقایسه قوی‌ای را در اختیار پژوهشگران قرار دهد.
- تغییر نزولی سطح عملیاتی: در این چارچوب پیاده‌سازی، امکان افزایش حالت عملیاتی سامانه از عادی به بحرانی وجود دارد اما سازوکاری برای بازگرداندن سامانه به حالت عادی تعییه نشده‌است. پیاده‌سازی چنین سازوکارهایی می‌تواند گام مهمی در حرکت به سوی صنعتی‌سازی سامانه‌های بحرانی مختلط باشد.
- افزایش تعداد هسته‌ها: چارچوب شبیه‌سازی‌شده در این پروژه از معماری مشخص و کلاس‌های مجزایی برخودار است و به راحتی می‌توان آن را برای پشتیبانی از پردازش‌های چند‌هسته‌ای که موضوعی مهم در زمینه‌ی تحقیقات سامانه‌های بحرانی مختلط هستند، گسترش داد.
- در نظر گرفتن سربار: در بررسی‌های خود اشاره کردیم که بیشتر الگوریتم‌های زمان‌بندی بحرانی مختلط، هزینه‌های ناشی از سربار سازوکارهای نظارتی در زمان اجرا را در نظر نمی‌گیرند. برای دقیق‌تر سازی عملکرد شبیه‌ساز، می‌توان هزینه‌ی سربار را نیز در آینده به آن اضافه کرد.

## ٧ - مراجع

- [١] S. K. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *ECRTS*, 2008.
- [٢] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium (RTSS)*, 2007.
- [٣] A. Burns and R. I. Davis, Mixed Criticality Systems - A Review (13th Edition), York: The University of York, 2022, p. 97 (total pages).
- [٤] S. K. Baruah, V. Bonifaci, G. d'Angelo, A. Marchetti-Spaccamela and S. van Der Ster, "Mixed-Criticality Scheduling of Sporadic Task Systems [DOI: 10.1007/978-3-642-23719-5\_47]," in *19th Annual European Symposium on Algorithms (ESA 2011)*, Saarbruecken, 2011.
- [٥] G. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Springer US, 2010 ,p. 426 (total pages.)
- [٦] S. K. Baruah, A. Burns and R. I. Davis, "Response-time analysis for mixed criticality systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2011.

- [٧] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow و L. Stougie, "Scheduling real-time mixed-criticality jobs", *35th Symposium on Mathematical Foundations of Computer Science 2010* ,
- [٨] Node.js, "Node.js - Run Javascript Everywhere," n.d. [Online]. Available: <https://nodejs.org/en>. [Accessed April 2024].
- [٩] Microsoft - Typescript, "TypeScript - Javascript with Syntax for Types.,," n.d. [Online]. Available: <https://typescriptlang.org/>. [Accessed April 2024].
- [١٠] Nashwaan, "xml-js," 2019. [Online]. Available: <https://github.com/nashwaan/xml-js>. [Accessed April 2024].
- [١١] JetBrains IDE, "Webstorm: The JavaScript and TypeScript IDE, by JetBrains," n.d. [Online]. Available: <https://www.jetbrains.com/webstorm/>. [Accessed April 2024].
- [١٢] Yarn Package Manager, "Yarn," n.d.. [Online]. Available: <https://yarnpkg.com/>. [Accessed April 2024].