بهبود قابلیت اطمینان سیستم‌های بلادرنگ با استفاده از بازیابی پویا

امیرحسین محمدی[1]، محسن انصاری[2]، علیرضا اجلالی[3]

[1] دانشجوی کارشناسی ارشد، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف
تهران، ایران
ahmohammadi@ce.sharif.edu,

[2] دانشجوی دکتری، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف
تهران، ایران
mansari@ce.sharif.edu,

[3] دانشیار، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف
تهران، ایران
ejlali@sharif.edu

چکیده

امروزه با توجه به اهمیت اشکال‌های گذرا[1] که از خطاهای نرم[2] و اشکال‌های دائمی[3] که از فرسودگی قطعات ناشی می شوند، بخش وسیعی از تحقیقات در حوزه‌ی طراحی سیستم‌های سخت افزاری و نرم افزاری به ارائه‌ی روش‌هایی برای مقابله با این دو اشکال[4] متمرکز شده است. اگر چه تاکنون متدها و الگوریتم‌های گسترده‌ای توسط محققان برای مقابله با این دو اشکال ارائه شده است، با این حال اغلب این الگوریتم‌ها فقط به مقابله با یکی از این دو اشکال می‌پردازند و یا الگوریتم‌های ساده‌ای برای تخصیص امکان بازیابی به تسک‌های شکست خورده هستند که به صورت کارآمدی از زمان باقی مانده[5] استفاده نمی‌کنند و سبب بهبود چشمگیر قابلیت اطمینان خطاهای نرم نمی‌شوند. در این مقاله به معرفی چارچوبی نرم افزاری جهت بهبود قابلیت اطمینان خطاهای نرم[6] به همراه قابلیت اطمینان طول عمر[7] سیستم، با توجه به محدودیت‌های سیستم‌های بلادرنگ[8] می‌پردازیم. ما یک تکنیک پویا جهت تخصیص امکان بازیابی[9] به تسک‌های شکست خورده[10] معرفی می‌کنیم که تضمین می‌کند به شرطی که زمان باقی مانده کافی باشد، همه‌ی تسک‌های شکست خورده را بازیابی کند. بر اساس این تکنیک، ما دو الگوریتم زمانبدی جهت بهبود سطح قابلیت اطمینان خطاهای نرم برای مجموعه‌ای از تسک‌ها با مشخصه‌های گوناگون پیشنهاد می‌کنیم. همچنین جهت بهبود قابلیت اطمینان طول عمر سیستم، از کاهش سطح فرکانس هسته‌های[11] پردازشی برای تسک‌های مناسب استفاده می‌کنیم که در نتیجه سبب کاهش فرسودگی سیسستم می شود. نتایج شبیه سازی نشان می‌دهد که چارچوب پیشنهاد شده در این مقاله، احتمال خرابی سیستم را بین ٪8 تا ٪73 درصد در مقایسه با تکنیک‌های موجود بهبود می‌بخشد.

کلمات کلیدی

قابلیت اطمینان خطاهای نرم، قابلیت اطمینان طول عمر، بازیابی پویا، سیستم‌های تعبیه شده‌ی بلادرنگ

1- مقدمه

امروزه تعداد قابل توجهی از سیستم‌های نهفته‌ی[12] بلادرنگ در سیستم‌های ایمنی بحران[13] مورد استفاده قرار می‌گیرند و اغلب این سیستم‌ها گران قیمت هستند و تعمیر و جایگزینی آن‌ها دشوار است. بنابراین قابلیت اطمینان خطاهای نرم (SER) که از اشکال‌های گذرا ناشی می شوند و قابلیت اطمینان طول عمر سیستم (LTR) که در اثر خطاهای دائمی به وجود می‌آیند، به عنوان جنبه‌های مهم در طراحی سیستم‌های سخت افزاری و نرم افزاری در نظر گرفته می‌شوند.

در حال حاضر تکنیک‌هایی جهت کاهش توان مصرفی، انرژی و دما وجود دارند که اغلب این تکنیک‌ها بر اساس کاهش سطح فرکانسی هسته‌های پردازشی[14] هستند و در نتیجه به بهبود LTR منجر می‌شوند، اما سطح SER را کاهش می‌دهد و بالعکس [1]. در این مقاله، ما بر روی بهبود حداکثری سطح SER سیستم تمرکز می‌کنیم

[1] Transient Faults
[2] Soft Errors
[3] Permanent Faults
[4] Fault
[55] Time Slack
[6] Soft Errors Reliability
[7] Life Time Reliability
[8] Real Time
[9] Dynamic Recovery Allocation
[10] Failed Tasks
[11] Core Frequencies
[12] Embedded Systems
[13] Safety Critical
[14] Core Frequency

و این در حالی است که سطح LTR سیستم از یک مقدار پیش فرض کاهش نمی‌یابد [2] و همچنین محدودیت‌های سیستم های بلادرنگ همچون ددلاین نیز در نظر گرفته می‌شوند. جهت بهبود حداکثری SER سیستم، ما به معرفی یک تکنیک بازیابی پویای نوین می‌پردازیم که Slack موجود را تقسیم می‌کند و تمام تسک های شکست خورده به شرط اینکه میزان Slack باقی مانده کافی باشد [4]، می‌توانند بازیابی شوند. یک تسک هنگام بازیابی از ابتدا اجرا می شود و ما فرض می‌کنیم همه‌ی تسک‌ها می‌توانند مجددا از ابتدا اجرا شوند. ما همچنین فرض می‌کنیم اشکال‌ها در پایان هر تسک شناسایی می‌شوند و از سربار محاسباتی عملیات تشخیص فالت‌ها صرف نظر می‌کنیم [11]. با توجه به اینکه اثربخشی تکنیک‌های بازیابی پویا به ترتیب اجرای تسک‌ها بستگی دارد، بنابراین ما به بررسی نحوه‌ی تاثیر زمانبدی تسک‌ها در سطح SER سیستم و به معرفی دو الگوریتم زمانبدی جهت بهبود سطح SER سیستم برای تسک‌ها با مشخصه‌های گوناگون می‌پردازیم.

ما به طراحی یک چارچوب جهت بهبود قابلیت اطمینان خطاهای نرم می‌پردازیم و آن را RIF می نامیم. RIF از دو بخش تشکیل شده است. هدف از طراحی بخش اول، افزایش سطح SER با استفاده از زمانبدی تسک‌ها (تعیین اولویت تسک‌ها) و بازیابی پویا است. اگر مقدار LTR از محدوده‌ی پیش فرض تعیین شده کمتر باشد، بخش دوم RIF به کار برده می‌شود تا LTR سیستم را با استفاده از کاهش سطح فرکانسی برای تسک‌های مناسب، افزایش دهد. جهت پیاده سازی این چارچوب، ما سه نکته‌ی اصلی خواهیم داشت. i) ما یک تکنیک تخصیص بازیابی پویای نوین پیشنهاد می‌کنیم که سطح SER سیستم را بهبود می‌بخشد. ii) ما به معرفی دو الگوریتم زمانبدی برای تسک‌ها با مشخصه‌های گوناگون، جهت بهبود سطح SER می‌پردازیم. الگوریتم اول از نظر محاسباتی کارآمد است و برای حالاتی که زمان اجرای تسک‌ها در یک مجموعه مشابه یکدیگر و Slack در دسترس کم است، مناسب است. الگوریتم دوم بسیار قدرتمند است و به صورت عمومی برای همه‌ی تسک‌ها مناسب است اما پیچیدگی زمانی بیشتری دارد. iii) توصیه می‌شود برای مجموعه تسک‌هایی که ویژگی‌های گفته شده در بخش ii را دارا هستند از الگوریتم اول که هزینه‌ی کمتری دارد، استفاده شود.

2- ساختار سیستم و فرمول بندی مسئله

در این بخش، ما در ابتدا به معرفی ساختار سیستم می‌پردازیم. سپس به معرفی و ترسیم ابعاد مختلف چارچوب طراحی شده خواهیم پرداخت.

A. ساختار تسک

در این پیاده سازی فرض می‌کنیم هر مجموعه از تسک‌ها[1]، شامل تسک‌های مستقل از هم می‌باشد. همچنین هسته‌های پردازشی از L سطح فرکانسی پشتیبانی می‌کنند. سطوح فرکانسی به صورت افزایشی مرتب شده‌اند، به این صورت که سطح l_L بیشترین سطح فرکانسی را دارد. ما فرض می‌کنیم که تسک‌ها در زمان طراحی[2] به هسته‌ها تخصیص می‌یابد و امکان انتقال تسک‌ها بین هسته‌ها وجود ندارد. همچنین ما فرض می‌کنیم که یک تسک و بازیابی آن به صورت مشابه بر روی یک هسته اجرا می‌شوند.

یک تسک، T_i، به صورت یک چندتایی $\{f_i(l_i), c_i(l_i), p_i\}$ مشخص می شود. که در این چندتایی $f_i(l_i)$ سطح فرکانسی هسته پردازشی و $c_i(l_i)$ بدترین زمان اجرای برای *تسک* T_i در سطح فرکانسی l_i است. p_i اولویت تسک T_i را تعیین می‌کند (هرچه این عدد بزرگتر باشد اولویت تسک بالاتر است) و اولویت یک تسک ترتیب اجرای آن تسک را مشخص می‌کند. هر تغییری در اولویت یک تسک قابل قبول است به شرط اینکه محدویت‌های سیستم‌های بلادرنگ را نقض نکند. اجرای تسک‌ها باید در یک ددلاین مشترک، D، به اتمام برسد. از Slack جهت اجرای مجدد تسک‌های شکست خورده استفاده می‌کنیم و با s نمایش داده می شود. جهت محاسبه‌ی Slack داریم:

$$D - \sum_{i=0}^{\pi} ci(li) \qquad (1)$$

B. قابلیت اطمینان خطاهای نرم و طول عمر

قابلیت اطمینان خطاهای نرم یک تسک به این معنی است که چقدر احتمال دارد که اجرای یک تسک به صورت موفقیت آمیز به اتمام برسد. برای تسک T_i که در سطح فرکانسی l_i اجرا می‌شود، $r_i^t(l_i)$ احتمال این است که هیچ خطای نرمی در طول اجرای تسک T_i رخ ندهد. مقدار $r_i^t(l_i)$ به دو پارامتر سطح فرکانسی (l_i) و زمان اجرای ($c_i(l_i)$) تسک T_i وابسته است و داریم[2] [9] [10]:

$$r_i^t(l_i) = e^{-\lambda(l_i)\frac{ci(li)}{fi(li)}} \qquad (2)$$

که $\lambda(l_i)$ نرخ برخورد اشکال به سیستم مورد نظر است و داریم:

$$\lambda(l_i) = \lambda_L 10^{\frac{d\times(1-fi(li))}{1-fi(l1)}} \qquad (3)$$

که l_1 کمترین سطح فرکانسی هسته‌های پردازشی است و λ_L میانگین نرخ اشکال است که به سیستم مورد نظر ما در زمان اجرای تسک و در بالاترین سطح فرکانسی برخورد می‌کند. d یک ثابت خاص سخت افزاری است که نرخ حساسیت سیستم را نسبت به اشکال در سطوح فرکانسی مختلف نشان می‌دهد.

قابلیت اطمینان طول عمر به چندین عامل فرسودگی بستگی دارد. فرسودگی ناشی از انتقال الکتریکی[3]، انتقال استرس[4] و تجزیه دی الکتریک وابسته به زمان[5] به طورقابل ملاحظه‌ای به دمای عملیاتی[6] بستگی دارد. همچنین فرسودگی‌های ناشی از سیکل حرارتی[7] نیز به

[1] Task Set
[2] Design Time
[3] Electromigration
[4] Stress Migration
[5] Time Dependent Dielectric Breakdown
[6] Operating Temperature
[7] Thermal Cycling

صورت تصاعدی به دامنه[1]، دوره[2] و ماکسیمم سیکل دمایی[3] وابسته است. بنابراین جهت بهبود LTR نیاز داریم دمای عملیاتی و تاثیرات سیکل حرارتی را کاهش دهیم.

C. فرمول بندی مسئله

با توجه به اهمیت پارامترهای SER، LTR و محدودیت‌های سیستم‌های بلادرنگ که در بخش‌های قبل به آن پرداختیم، ما برآنیم تا مسئله‌ی ماکسیمم کردن سطح SER را برای هر هسته‌ی پردازشی، R_{sys}، حل کنیم در عین حال محدودیت‌های سیستم‌های بلادرنگ و LTR را در نظر بگیریم:

$$\sum_{Ti\in\pi} ci(li)\leq D, \qquad (3)$$

$$MTTF \geq MTTF_{TH}, \qquad (4)$$

به طوریکه پارامتر $MTTF_{TH}$، مینیمم $MTTF$ است که سیستم باید به آن برسد و پارامتر π تعداد تسک‌هایی هستند که بر روی یک هسته اجرا می‌شوند. R_{sys} پارامتری است که احتمال اجرای موفقیت آمیز همه‌ی تسک‌ها را مشخص می‌کند. یک تسک موفقیت آمیز در نظر گرفته می‌شود، اگر در اولین اجرا یا در دومین اجرای خود (وقتی با اشکال مواجه شده و بازیابی می شود) به صورت موفقیت آمیز به اتمام برسد [7]. بنابراین محاسبه‌ی R_{sys} علاوه بر SER هر تسک، به تکنیک بازیابی نیز وابسته است. در ادامه به بحث در رابطه با تکنیک بازیابی پویا که توسط ما پیشنهاد می‌شود، خواهیم پرداخت. توجه داشته باشید که ما فرض می‌کنیم که یک تسک و بازیابی آن به طور کامل بر روی یک هسته اجرا می‌شوند.

D. مروری بر ساختار مدل پیشنهادی

ما یک چارچوب (RIF) جهت بهبود قابلیت اطمینان خطاهای نرم و حل مسائلی که در بخش‌های پیشین اشاره کردیم، پیشنهاد می‌کنیم. RIF از دو بخش تشکیل شده است. یکی از این دو بخش بر روی بهبود افزایش سطح SER تمرکز دارد و دیگری محدودیت‌های سیستم‌های بلادرنگ و LTR را در نظر می‌گیرد. به عبارت دیگر جهت ماکسیمم کردن سطح SER، بخش اول سیستم RIF، تسک‌ها را در بالاترین سطح فرکانسی اجرا می‌کند و SER سیستم را با استفاده از بازیابی پویا و زمانبدی تسک‌ها بهبود می‌بخشد. ما در ابتدا به نحوه‌ی محاسبه‌ی سطح SER با توجه به تکنیک بازیابی پویا که توسط ما پیشنهاد شده است خواهیم پرداخت و سپس به تاثیر زمانبندی تسک‌ها بر سطح SER سیستم می‌پردازیم. ما همچنین به توصیف یک الگوریتم زمانبدی کارآمد برای تسک‌هایی با ویژگی‌های مشخص و یک الگوریتم قدرتمندتر و عمومی برای تسک‌های گوناگون می‌پردازیم. بخش دوم RIF وظیفه ی بررسی LTR سیستم را دارد. اگر LTR سیستم از مقدار پیش فرض کمتر شود، سطح فرکانسی تسک‌ها با درجه اولویت کمتر و تسک‌هایی که توان زیادی مصرف می‌کنند، کاهش می‌یابد تا محدودیت‌های سیستم‌های بلادرنگ و سطح LTR سیستم رعایت شود.

3- بهبود قابلیت اطمینان خطاهای نرم

در این بخش، به معرفی تکنیک بازیابی پویا که توسط ما طراحی شده است و بررسی تاثیر زمانبدی تسک‌ها بر روی سطح SER سیستم می‌پردازیم. ما اجازه می‌دهیم تسک‌ها در بالاترین سطح فرکانسی اجرا شوند و سپس با زمانبدی تسک‌ها سطح SER سیستم را بهبود می‌دهیم.

A. تکنیک بازیابی پویا

ما یک رویکرد جدید برای تخصیص امکان بازیابی به تسک‌های شکست خورده ارائه می‌کنیم. بر اساس تکنیک ما، Slack در دسترس بین تمام تسک‌های شکست خورده تقسیم می شود و به صورت پویا و بر اساس متد FCFS[4] به تسک‌ها اختصاص می‌یابد. امکان بازیابی به تسک T_i تخصیص می‌یابد اگر مقدار Slack باقی مانده از $c_i(l_L)$کوچکتر نباشد. بازیابی‌ها باید در بالاترین سطح فرکانسی اجرا شود. اگرچه ممکن است یک تسک بازیابی شده مجددا شکست بخورد اما از آنجایی که احتمال وقوع آن بسیار کم است ما به هر تسک یک امکان بازیابی اختصاص می‌دهیم تا از مصرف بیش از حد Slack توسط یک تسک جلوگیری کنیم. از آن جایی که امکان بازیابی به یک تسک زمانی اختصاص داده می شود که مقدار Slack کافی باشد، بنابراین بازیابی‌ها محدودیت‌های سیستم‌های بلادرنگ را نقض نمی‌کند [5]. از آن جایی که اجرای یک تسک با اولویت بالاتر و مصرف Slack توسط آن می‌تواند بر روی اختصاص امکان بازیابی به تسک‌های با اولویت کمتر اثر بگذارد بنابراین زمانبدی تسک ها به صورت مستقیم بر سطح SER تاثیر دارد.

فرض کنید می‌خواهیم سطح SER را برای یک زمانبد، S محاسبه کنیم. برای یک زمانبد، پارامتر $r_i^s(S)$، احتمال اینکه یک تسک بازیابی دارد و بازیابی آن موفقیت آمیز است را تعیین می‌کند. بنابراین احتمال اینکه یک تسک موفقیت امیز به پایان است برابر است با:

$$r_i(S) = 1 - (1 - r_i^t)(1 - r_i^s(S)) \qquad (5)$$

بنابراین برای محاسبه‌ی سطح SER سیستم از رابطه بالا داریم:

$$R_{sys}(S) = \prod_{i=1}^{n}\{1 - (1 - r_i^t)(1 - r_i^s(S))\} \qquad (6)$$

$R_{sys}(S)$، سطح SER سیستم را بر اساس زمانبد S تعیین می‌کند. جهت محاسبه‌ی $R_{sys}(S)$، کلیدی ترین پارامتر $r_i^s(S)$ است. ما از یک مفهوم با عنوان الگوی اجرایی برای محاسبه‌ی $r_i^s(S)$ استفاده می‌کنیم. زمانبند $S=\{ T_1, \ldots, T_n \}$ را در نظر بگیرید که در آن تسک T_i نسبت به T_{i+1} از اولویت بالاتری برخوردار است. ما از الگوی اجرایی P_i که تسک‌های شکست خورده و موفقیت آمیز قبل از تسک T_i را

[1] Amplitude
[2] Period
[3] Cycle Maximum Temperature
[4] First-Come, First Serve

مشخص می‌کند، استفاده می‌کنیم. در یک الگوی اجرایی، پارامتر T_k^+ تسک‌های موفقیت آمیز را از بین مجموعه تسک‌های T_k مشخص می‌کند و T_k^-، تسک‌های شکست خورده را مشخص می‌کند. به دلیل اینکه یک الگوی اجرایی از $i-1$ تسک تشکیل شده است، بنابراین ما 2^{i-1} الگوی اجرایی برای تسک T_i خواهیم داشت و پارامتر $P_{i,j}$، j امین الگوی اجرایی را مشخص می‌کند. اگر ما مقدار زمان مصرف شده توسط تسک‌های بازیابی شده را که با $T(P_{i,j})$ مشخص می‌شود بدانیم، مقدار $r_i^s(S)$ قابل محاسبه است. همچنین با استفاده از پارامتر $Prob(P_{i,j})$، احتمال وقوع هر الگوی اجرایی را محاسبه می‌کنیم. فرض کنید ما m الگوی اجرایی داریم که از رابطه‌ی زیر پیروی می‌کنند:

$$T(P_{i,j}) + c_i(l_L) \le s \qquad (7)$$

همچنین می‌دانیم s مقدار Slack تقسیم شده بین تسک‌ها است. بنابراین برای محاسبه‌ی $r_i^s(S)$ داریم:

$$r_i^s(S) = \begin{cases} r_i^t \times \sum_{j=1}^{m} Prob(P_{i,j}), & if\ m > 0 \\ 0 \quad , & otherwise \end{cases} \qquad (8)$$

اگر m=0 باشد به این معنی است که T_i نمی‌تواند بازیابی شود. $T(P_{i,j})$ و $Prob(P_{i,j})$ به عنوان دو پارامتر کلیدی جهت محاسبه‌ی $r_i^s(S)$ در هر الگوی اجرایی مطرح می‌شوند. همانطور که در قسمت‌های قبل هم اشاره شد، یک تسک T_i، زمانی می‌تواند بازیابی شود که مقدار باقی مانده‌ی Slack بزرگتر از $c_i(l_L)$ باشد. بنابراین ممکن است یک تسک که زمان اجرایی زیادی دارد، نتواند از Slack باقی مانده استفاده کند [4] [6]. ما تسک‌های شکست خورده در هر $P_{i,j}$ پیدا کرده و مقدار $T(P_{i,j})$ را برای آن محاسبه می‌کنیم. ما در ابتدا $T(P_{i,j}) = 0$ را مقداردهی می‌کنیم. سپس برای هر تسک شکست خورده‌ی T_k در $P_{i,j}$، اگر $c_k(l_L) \le s$ باشد، آنگاه $T(P_{i,j}) = T(P_{i,j}) + c_k(l_L)$ و $s = s - c_k(l_L)$ به روز رسانی می شود. $Prob(P_{i,j})$ به سطح SER هر تسک در یک الگوی اجرایی بستگی دارد، بنابراین داریم:

$$Prob(P_{i,j}) = \prod_{T_k^+ \in P_{i,j}} T_k^t \times \prod_{T_k^- \in P_{i,j}} 1 - T_k^t \qquad (9)$$

بر اساس رابطه‌ی (6)، زمانبدی تسک‌ها به صورت مستقیم بر سطح SER سیستم تاثیر می‌گذارد. ما در ادامه دو الگوریتم زمانبدی جهت بهبود سطح SER سیستم و برای مجموعه‌ی تسک‌ها با مشخصات متفاوت ارائه می‌کنیم.

B. یک الگوریتم زمانبدی کارآمد

ما یک الگوریتم زمانبدی کارآمد جهت بهبود قابلیت اطمینان خطاهای نرم (ERIS) معرفی می‌کنیم. جهت بهبود سطح SER، تسک‌ها باید ویژگی‌های معینی داشته باشند.

نکته‌ی قابل توجه در زمانبدی تسک‌ها این است که همیشه بازیابی بیشتر تسک‌ها به افزایش سطح SER سیستم منجر نمی شود. برای مثال اگر یک زمانبد، تسک‌های زیادی با زمان اجرای کوتاه را بازیابی کند، مقدار Slack باقی مانده کاهش می‌یابد و امکان بازیابی تسک‌های با زمان اجرای طولانی وجود نخواهد داشت و در نهایت سطح SER سیستم به طور کلی کاهش می‌یابد. فقط در صورتی زمانبدی یک مجموعه تسک و بازیابی زیاد تسک‌ها منجر به افزایش سطح SER می‌شود که دو شرط زیر برقرار باشد:

فرضیه‌ی 1

$$s < c_{max} \qquad (i$$

$$(n-1)c_{min} \ge 2c_{max} \qquad (ii$$

بنابراین بر اساس *فرضیه‌ی 1*، زمانبدی تسک‌ها به افزایش بازیابی‌ها و همچنین افزایش سطح SER سیستم منجر می‌شود. c_{min} و c_{max} به ترتیب کمترین و بیشترین زمان اجرا در بین تسک‌ها (در بالاترین سطح فرکانسی هسته‌های پردازشی) هستند. مشاهدات نشان می‌دهد که یک تسک که اولویت بیشتری دارد و زودتر اجرا می‌شود، با احتمال بیشتری می‌تواند بازیابی شود و این مسئله به این خاطر است که Slack کمتری در این حالت توسط تسک‌های پیشین مصرف شده است. به عبارت دیگر جهت حفظ Slack برای استفاده‌ی تسک‌های بعدی، تسک‌های با اولویت بالاتر باید زمان اجرای کمتری داشته باشند. بنابراین ما یک الگوریتم زمانبدی کارآمد (ERIS) جهت تعیین اولویت تسک‌ها با توجه به زمان اجرای آن‌ها معرفی می‌کنیم. ما تسک‌ها را بر اساس *فرضیه‌ی 1* در نظر می‌گیریم و این تسک‌ها بر اساس ERIS زمانبدی می شود. به طور مثال اگر دو تسک T_i و T_j را در نظر بگیریم و اگر $c_i(l_L) > c_j(l_L)$ باشد، آنگاه $p_i < p_j$ خواهد بود و سطح SER سیستم ماکسیمم خواهد شد.

ما یک متد جدید برای محاسبه‌ی سطح SER سیستم در زمان شبه چند جمله‌ای، برای تسک‌هایی که از *فرضیه‌ی 1* پیروی می‌کنند و بر اساس ERIS زمانبدی می شوند، طراحی می‌کنیم. این متد سربار محاسباتی پیدا کردن الگوهای اجرایی که از رابطه‌ی (7) پیروی می‌کنند، کاهش می‌دهد.

فرض کنید یک زمانبد $S = \{T_1, \dots, T_n\}$ شامل n تسک است، و داریم $p_i > p_{i+1}$ و $c_i(l_L) < c_{i+1}(l_L)$. برای تسک T_i، ما دو مفهوم معرفی می‌کنیم: i) مجموعه ی سنگین[1]، Ω_i^+، یک زیر مجموعه از تسک‌های شکست خورده در $S = \{T_1, \dots, T_{i-1}\}$ است که Slack مورد نیاز برای بازیابی این تسک ها از $s - c_i(l_L)$ بیشتر است. ii) مجموعه‌ی سبک[2]، Ω_i^-، یک زیر مجموعه از تسک‌های شکست خورده در $S = \{T_1, \dots, T_{i-1}\}$ است که Slack مورد نیاز برای بازیابی این تسک‌ها از $s - c_i(l_L)$ کوچکتر و مساوی است. تسک‌های موجود در مجموعه‌های Ω_i^+ و Ω_i^- به صورت کاهش اولویت مرتب شده‌اند. پارامتر $\Omega_{i,j}^-$، jامین مجموعه‌ی سبک برای تسک T_i مشخص

[1] Heavy Set

[2] Light Set

می‌کند. اگر $\Omega_{i,j}^{-} = \{T_1, T_2\}$ را در نظر بگیریم، به این معنی است که Slack مورد نیاز برای برای بازیابی T_1 و T_2 از $s - c_i(l_L)$ کمتر یا مساوی است. بنابراین بر اساس این مفهوم، مجموعه‌ی سبک با رابطه‌ی بیان شده برای الگوی اجرایی در رابطه ی (7) تطابق دارد و می‌توانیم از آن برای محاسبه‌ی سطح SER سیستم استفاده کنیم. در ادامه به بیان سه قضیه‌ی کاربردی جهت یافتن مجموعه‌ی سبک می‌پردازیم.

قضیه 1: بر اساس ERIS، مجموعه‌ی Ω_i یک مجموعه‌ی سنگین برای T_i است اگر و تنها اگر $\sum_{T_j \in \Omega_i} c_j(l_L) > s - c_i(l_L)$.

قضیه 2: اگر $\Omega_{i,j}^{+}$ یک مجموعه‌ی سنگین از T_i باشد، $\Omega_{i,k}^{+}$ نیز یک مجموعه‌ی سنگین است، اگر و تنها اگر $\Omega_{i,k}^{+}$ یک ابر مجموعه[1] برای $\Omega_{i,j}^{+}$ باشد $\Omega_{i,j}^{+} < \Omega_{i,k}^{+}$.

قضیه 3: بر اساس ERIS، اگر $\Omega_{i,j}^{+}$ یک مجموعه‌ی سنگین برای تسک T_i باشد، آنگاه $\Omega_{i,j}^{+}$ برای T_{i+1} نیز یک مجموعه‌ی سنگین است.

برای تسک T_i، ما همه‌ی مجموعه‌های سبک مربوط به آن را به گروه‌هایی تقسیم می‌کنیم. $G_{i,k}$، kامین گروه از مجموعه‌های سبک T_i است که دقیقا k تسک سبک دارند. بر اساس این مفهوم برای هر تسک T_i، i گروه $G_{i,0} \dots G_{i,i-1}$ وجود دارد. بر اساس قضیه‌ی 1 و 3 و توصیف مفهوم $G_{i,k}$، در ادامه به روشی جهت یافت همه‌ی مجموعه‌های سبک برای هر تسک می‌پردازیم که در نهایت به ماکسیمم شدن سطح SER کمک می‌کند.

فرضیه‌ی 2

برای مجموعه تسک‌هایی که از ویژگی‌های یاد شده در *فرضیه‌ی1* پیروی می‌کنند و بر اساس ERIS زمانبدی می‌شوند، اگر یک مجموعه‌ی سبک، Ω_i^{-}، در داخل گروه $G_{i-1,k}$ وجود داشته باشد و از رابطه‌ی $\sum_{T_i \in \Omega^-} c_j(l_L) \leq s - c_i(l_L)$ پیروی کند، آنگاه Ω^-، یک مجموعه‌ی سبک برای گروه $G_{i,k}$ نیز هست. اگر Ω^- یک مجموعه‌ی سبک برای $G_{i-1,k-1}$ باشد، آنگاه یک مجموعه سبک برای $G_{i,k}$ نیز خواهد بود به شرط اینکه $c_{i-1}(l_L) - \sum_{T_i \in \Omega^-} c_j(l_L) \leq s - c_i(l_L)$ باشد. بر اساس *قضیه‌ی 2* ما تمام مجموعه‌های سبک برای هر تسک را به صورت تکراری محاسبه می‌شود.

جزئیات متد ما در *الگوریتم 1* نشان داده شده است. ما در ابتدا $G_{i,0}$ برای هر تسک مقداردهی می‌کنیم در خط 2-8. اگر $c_i(l_L) \leq s$ با شد آنگاه $G_{i,0}$ با $\{0\}$مقداردهی می کنیم که به این معنی ا ست که تسک T_i می تواند بازیابی شود اگر مقدار Slack کافی باشد. اگر مقدار Slack کافی نباشد آنگاه $G_{i,0} = 0$، به این معنی است یک تسک نمی‌تواند بازیابی شود. ما $G_{i,k}$ را برای تسک‌های مختلف محاسبه می‌کنیم در خط 9-28. بر اساس *قضیه‌ی 2*، $G_{i,k}$ در ابتدا با 0 مقداردهی می شود (در خط 11) و مجموعه‌های سبک به $G_{i,k}$ ا ضافه می شود در خط 12-22. برای هر مجموعه‌ی سبک در $G_{i-1,k}$، اگر همچنان یک مجموعه‌ی سبک برای T_i باشد، آنگاه به $G_{i,k}$ اضافه می‌شود (12-16). به طور مشابه، برای هر مجموعه‌ی سبک در $G_{i-1,k-1}$، تعیین می‌کنیم که به شرطی T_{i-1} اضافه می‌شود که مجموعه همچنان سبک باقی بماند(خط 17-22). بر ا ساس *قضیه 2*، اگر $G_{i,k} = 0$، گروه های $G_{i,k+1}$ تا $G_{i,i+1}$ را با صفر مقداردهی می کنیم (23-26). در آخر ما تمامی مجموعه‌های سبک $G_{i,0}, \dots, G_{i,i-1}$ را بر می گردانیم (خط 29).

C. یک الگوریتم عمومی جهت زمانبدی

همانطور که گفته شد، ERIS یک الگوریتم زمانبدی کارآمد و موثر برای تسک‌هایی ا ست که از شرایط موجود در *فرضیه‌ی 1* پیروی می‌کنند. بنابراین برای زمانبدی تسک‌هایی که ویژگی‌های گوناگونی دارند، ما به یک الگوریتم زمانبدی دیگری نیاز داریم که به صورت عمومی مورد استفاده قرار می‌گیرد. در این بخش ما به معرفی یک الگوریتم ز مان بدی عمومی (GRIS) ج هت بهبود قابل یت اطمی نان خطاهای نرم برای تسک‌های مختلف (که می‌توانند هر مقدار دلخواهی داشته باشند) می‌پردازیم. GRIS تضمین می‌کند که سطح SER سیستم همیشه بالاتر از تکنیک‌های بازیابی استاتیک است. GRIS در ابتدا بهینه‌ترین راحل برای تخصیص بازیابی ایستا[2] تسک‌ها یعنی مجموعه‌ی Φ را می‌یابد. سپس اولویت تسک‌های درون Φ را جهت بهبود سطح SER سیستم افزایش می‌دهد.

الگوریتم 1 یافتن مجموعه‌های سبک

```
1:  procedure FIND_SET(S = {τ_1, τ_2, ..., τ_n})
2:      for task τ_i in {τ_1, τ_2, ..., τ_n} do
3:          if c_i(l_L) ≤ s then
4:              G_{i,0} = {∅}
5:          else
6:              G_{i,0} = ∅
7:          end if
8:      end for
9:      for task τ_i in {τ_2, τ_3, ..., τ_n} do
10:         for k in {1, 2, ..., i − 1} do
11:             G_{i,k} = ∅
12:             for light set Ω^− in G_{i−1,k} do
13:                 if Σ_{τ_j∈Ω^−}(c_j(l_L)) ≤ s − c_i(l_L) then
14:                     G_{i,k} = {G_{i,k} ∪ Ω^−}
15:                 end if
16:             end for
17:             for light set Ω^− in G_{i−1,k−1} do
18:                 if c_{i−1}(l_L) + Σ_{τ_j∈Ω^−}(c_j(l_L)) ≤ s − c_i(l_L) then
19:                     Ω^− = {Ω^− ∪ τ_{i−1}}
20:                     G_{i,k} = {G_{i,k} ∪ Ω^−}
21:                 end if
22:             end for
23:             if G_{i,k} = ∅ then
24:                 G_{i,k+1} = ··· = G_{i,i−1} = ∅
25:                 break
26:             end if
27:         end for
28:     end for
29:     return {G_{i,0}, ..., G_{i,i−1}} for i in {1, 2, ..., n}
30: end procedure
```

مسئله‌ی تخصیص استاتیک بازیابی تسک‌ها به عنوان کلیدی‌ترین بخش در GRIS مطرح است. مسئله‌ی تخصیص بازیابی ایستای

[1] Super Set

[2] Static

تسک‌ها به عنوان یک گونه از مسئله‌ی *کوله پشتی*[1] است که با استفاده از روش‌های برنامه نویسی پویا قابل پیاده سازی است. فرض کنید $\Phi\{i, s'\}$ یک مجموعه از تسک‌هایی است که بیشترین سطح SER سیستم را بدست می‌آورند و بعضی از تسک‌های $\{T_1, T_2, \dots, T_i\}$ می‌توانند بازیابی شوند، اگر Slack مورد نیاز آن‌ها از s' کمتر یا مساوی باشند.

بنابراین بر اساس ساختار و ویژگی‌های مسئله ما، یک برنامه‌ی پویا بر اساس مسئله‌ی *کوله پشتی* می‌تواند بهترین راه حل برای پیدا کردن $\Phi\{n, s\}$ با شد. بر اساس راه حلی که برای تخصیص بازیابی ایستای تسک‌ها ارائه شد، تسک‌های داخل مجموعه‌ی Φ، دارای اولویت بیشتری نسبت به تسک‌هایی که داخل Φ نیستند، دارد. برای تسک‌های T_i و T_j که در مجموعه‌ی Φ قرار دارند، اولویت یک تسک به گونه‌ای تعیین می شود که اگر $c_i(l_L) > c_j(l_L)$ باشد آنگاه $p_i < p_j$. بر اساس تخصیص بازیابی پویا، این الکوریتم زمانبدی تضمین می‌کند همه‌ی تسک‌های موجود در مجموعه‌ی Φ بتوانند بازیابی شوند. همچنین اگر تعدادی از تسک‌های داخل مجموعه‌ی Φ به صورت موفقیت آمیز اجرا شوند، تسک های خارج از مجموعه‌ی Φ نیز می‌توانند بازیابی شوند. از این رو ، GRIS به یک SER بالاتر در سطح سیستم نسبت به راه حل بهینه برای مسئله تخصیص بازیابی ایستا دست می‌یابد.

4- حفظ قابلیت اطمینان طول عمر سیستم

جهت بهبود سطح SER سیستم، ناچاریم تسک‌ها را در بالاترین سطح فرکانسی اجرا کنیم. بنابراین ممکن است محدوده‌ی LTR سیستم که در رابطه‌ی (4) بیان کردیم، نقض شود. برای حل این مشکل، ما پیشنهاد می‌کنیم که سطح فرکانس هسته‌های پرداز شی برای تسک ها را کاهش بدهیم تا محدودیت های مربوط به LTR سیستم رعایت شود. کاهش سطح فرکانسی هسته‌های پرداز شی برای تسک‌ها منجر به کاهش توان و دمای عملیاتی می شود، اما به نوبه خود ممکن است الزامات زمان بندی تسک‌ها را نقض کرده و میزان ورود اشکال‌های گذرا را افزایش دهد (براساس رابطه‌ی (2)). بنابراین ما به مصالحه بین حفظ LTR سیستم و افزایش سطح SER نیاز داریم. ما معرفی یک روش ابتکاری جهت کاهش فرکانس هسته ها برای تسک‌های مناسب می‌پردازیم. توجه داشته باشید که اگرچه ما سطح فرکانسی هسته‌ها را برای تسک‌های مناسب کاهش می‌دهیم، اما بازیابی‌ها همیشه در بالاترین سطح فرکانسی اجرا می‌شود. به دلیل اینکه احتمال شکست یک تسک بسیار کم است، اجرای بازایابی یک تسک در بالاترین سطح فرکانسی هسته‌های پرداز شی، به طور قابل توجهی در سطح LTR سیستم در طولانی مدت ندارد.

طبق فرضیات ما، LTR یک سیستم از دو پارامتر دمای عملیاتی و سیکل حرارتی تاثیر می‌گیرد. کاهش فرکانس هسته‌ها برای اجرای تسک‌ها در کاهش دمای عملیاتی موثر است اما ممکن است سیکل حرارتی ایجاد کند. اگر فرکانس تسک T_i از فرکانس دو تسک T_{i-1} و T_{i+1} بیشتر یا کمتر باشد، می‌تواند سیکل حرارتی بیشتری تولید کند. بنابراین ما می‌خواهیم فرکانس هسته‌ها رابرای تسک‌های مناسب کاهش دهیم، درحالی که از سیکل حرراتی بیشتر جلوگیری می‌کنیم.

کاهش سطح فرکانسی هسته‌ها برای تسک‌ها می‌تواند منجر به افزایش زمان اجرای و کاهش سطح SER و Slack در دسترس شود. بنابراین اجرای تسک ها در نهایت نرخ خرابی بیشتری خواهد داشت و Slack بیشتری برای بازیابی تسک‌ها مصرف می‌شود. از آنجایی که یک تسک با اولویت بالا پیش از تسک‌های دیگر Slack در دسترس را برای بازیابی مصرف می‌کند و ممکن است بر قابلیت اطمینان تسک‌ها با اولویت پایین تأثیر بگذارد، ما این اصل کلی را می‌پذیریم که تسک‌های با اولویت بالا در بالاترین سطح فرکانسی اجرا می شوند در حالی که به سمت تسک‌های با اولویت کمتر کاهش فرکانس می‌یابند. ما روشی را برای انتخاب یک تسک برای کاهش فرکانس هسته آن‌ها ارائه می‌کنیم [8]، به گونه ای که انجام این کار صرفه جویی در مصرف انرژی را به حداکثر و تأثیر آن بر قابلیت اطمینان سایر تسک‌ها را به حداقل می‌رساند. ما یک پارامتر تحت عنوان *نسبت انرژی مصرفی-زمان* $R_i(l_i = j, l_i = k)$ برای تسک T_i وقتی که فرکانس هسته از j امین سطح به k امین سطح تغییر می کند:

$$R_i(l_i = j, l_i = k) = \frac{\rho_i(l_i = j) - \rho_i(l_i = k)}{c_i(l_i = c) - c_i(l_i = j)} \qquad (10)$$

که $\rho_i(l_i = j)$ و $c_i(l_i = j)$ به ترتیب میزان توان مصرفی و بدترین زمان اجرا برای تسک T_i است زمانی که در j امین سطح فرکانسی اجرا می شود. برای یک زمانبد $S = \{T_1, \dots, T_n\}$ که T_1 بیشترین اولویت را دارد و T_n کمترین اولویت دارد، ما فرکانس تسک ها را برای تسک‌های مناسب کاهش می‌دهیم (همانطور که در *الگوریتم 2* می‌بینید). ما به صورت متناوب فرکانس هسته را برای تسک ها کاهش می‌دهیم تا زمانی که *MTTF* بزرگ تر از مقدار *MTTF* پیش فرض بشود (در خط 3-11). ما در هر بار تکرار، یک تسک را انتخاب می‌کنیم و فرکانس هسته‌ی آن تسک را یک سطح کاهش می‌دهیم تا به بالاترین مقدار R دست یابد و محدودیت‌های ددلاین را نقض نکند. در همین حال، برای جلوگیری از ایجاد سیکل حرارتی بیشتر، ما فقط تسکی را انتخاب می‌کنیم که دارای فرکانس بالاتر است نسبت به تسک‌هایی که اولویت کمتری دارند $f_i > f_{i-1}$ (در خط 5). بعد از اینکه سطح فرکانسی هسته تسک انتخاب شده کاهش یافت، مقدار *MTTF* و s مجددا محاسبه می‌شود (در خط 10).

[1] Knapsack

الگوریتم 2 کاهش سطح فرکانسی

```
procedure FREQ_RED(S)
    ℜmax = 0, τt = τn
    while MTTFsys < MTTFTH do
        for τi ∈ {τ1, ..., τn−1} do
            if s > ci(li − 1) − ci(li) and ℜi(li, li − 1) > ℜmax
  and fi > fi+1 then
                ℜmax = ℜi(li, li − 1)
                τt = τi
            end if
        end for
        ft(lt) = ft(lt − 1) and update s and MTTFsys
    end while
end procedure
```

5- ارزیابی و شبیه سازی

جهت پیاده سازی الگوریتم‌های معرفی شده در این مقاله و طراحی چارچوب RIF، از زبان برنامه نویسی پایتون استفاده می‌کنیم. جهت شبیه سازی الگوریتم‌های پیاده سازی شده، ما به طراحی مجموعه تسک‌هایی بر اساس ساختار اشاره شده در این مقاله می‌پردازیم. در ادامه به توصیف روند پیاده سازی و شبیه سازی الگوریتم‌های معرفی شده در این مقاله خواهیم پرداخت.

A. پیاده سازی الگوریتم ERIS

جهت پیاده سازی این الگوریتم، همانطور که اشاره شد از زبان برنامه نویسی پایتون بهره می‌بریم. همانطور که گفته شد جهت شبیه سازی الگوریتم‌های پیاده سازی شده، به طراحی مجموعه تسک‌ها می‌پردازیم. با توجه به اینکه جهت شبیه سازی الگوریتم در این مقاله هر مجموعه تسک شامل 5 تسک در نظر گرفته شده است. بنابراین ما نیز به صورت پیش فرض مجموعه‌هایی با 5 تسک ایجاد می‌کنیم که هم تعداد این مجموعه‌ها و تعداد تسک‌ها در الگوریتمی که توسط ما پیاده سازی شده است، قابل تغییر است. همچنین طبق فرض مقاله، زمان این تسک‌ها بین بازه‌ی 0.75 تا 1.25 قرار داد و همانطور که پیش تر توضیح داده شد (i و ii)، الگوریتم ERIS برای تسک‌های با ویژگی‌های مشخص قابل استفاده است. ساختار الگوریتم ERIS بر اساس یافتن بهترین الگوی اجرایی (زمانبد) پیش از iامین تسک یک مجموعه است. بنابراین ما برای محاسبه‌ی قابلیت اطمینان سیستم، نیازمندیم تا تمامی الگوی‌های اجرایی (که از شرط‌های گفته شده در مقاله پیروی می‌کنند) پیش از تسک iام در یک مجموعه از تسک‌ها بیابیم. بنابراین ما به ایجاد قطعه کدی می پردازیم که بر اساس یک ساختار پویا، تمامی الگوهای اجرایی پیش از تسک i را پیدا می‌کنم. هر الگوی اجرایی شامل گروه‌هایی است. همچنین در این پیاده سازی هر الگوی اجرایی شامل دو مجموعه‌ی تسک‌های شکست خورده و تسک‌های موفقیت آمیز است. جهت محاسبه‌ی احتمال وقوع یک الگوی اجرایی، ما به پیاده سازی یک تابع جهت محاسبه این احتمال بر اساس تسک‌های موفقیت آمیز و شکست خورده (مطابق رابطه‌ی (9)) می‌پردازیم. ما احتمال وقوع تمامی الگوهای اجرایی که از رابطه‌ی (7) پیروی می‌کنند و پیش از تسک i قرار دارند محاسبه می‌کنیم. سپس برای محاسبه قابلیت اطمینان یک زمانبد که بر اساس تسک‌های پیش از تسک i طراحی شده است، به پیاده سازی تابع جهت محاسبه‌ی قابلیت اطمینان یک زمانبد بر اساس رابطه‌ی (8) می‌پردازیم. جهت محاسبه‌ی قابلیت اطمینان یک تسک طبق رابطه‌ی (2) و نرخ خرابی سیستم (3) به پیاده سازی توابعی جهت محاسبه‌ی این دو پارامتر می‌پردازیم. همچنین با توجه به مقاله، مقادیر d و λ_L را به صورت پیش فرض به ترتیب 3 و 10^{-6} در نظر می‌گیریم [10]. همچنین برای محاسبه‌ی قابلیت اطمینان سیستم به پیاده سازی تابعی بر اساس رابطه‌ی (5) و (6) می‌پردازیم. بر اساس این شبیه سازی و مجموعه تسک‌های تولید شده (مجموعه های که شامل 5 تسک هستند) و بازه‌ی زمانی گفته شده، احتمال خرابی سیستم به ازای 1000 مجموعه تسک و صورت میانگین 10^{-3} بدست آمد.

B. پیاده سازی الگوریتم GRIS

دومین الگوریتم که در این مقاله جهت زمانبدی تسک‌ها معرفی شده است، الگوریتم GRIS است. همانطور که پیش تر توضیح داده شد، GRIS یک الگوریتم زمانبدی عمومی برای تسک‌های با ویژگی‌های گوناگون است. در این مقاله به جزئیات پیاده سازی این الگوریتم اشاره‌ای نشده است، با این حال ما با توجه به فرضیات مقاله به پیاده سازی این الگوریتم می‌پردازیم. این الگوریتم بر اساس مسئله‌ی کوله پشتی طراحی شده است. بنابراین برای پیاده سازی این الگوریتم، ما از ایده‌ی برنامه نویسی پویا و مسئله‌ی کوله پشتی استفاده می‌کنیم. ما به پیاده سازی تابعی می‌پردازیم که بهترین زیر مجموعه از یک مجموعه تسک را پیدا می‌کند که بالاترین اولویت را دارند و زمان مورد نیاز برای بازیابی آن‌ها از Slack کوچکتر یا مساوی است. اگر همه یا بخشی از تسک‌های موجود در این مجموعه با موفقیت اجرا شوند، مجددا از بین تسک های موجود یک زیر مجموعه که بالاترین اولویت را دارند و زمان مورد نیاز برای بازیابی آن‌ها کوچکتر یا مساوی باشند، انتخاب می‌کنیم. این تابع بارها تکرار می‌شود تا زمانی که نتوانیم زیر مجموعه‌ای از تسک‌ها بیابیم که زمان مورد نیاز برای بازیابی آن‌ها از Slack موجود کوچکتر باشد یا اینکه تمامی تسک‌ها بتوانند بر اساس Slack موجود بازیابی و اجرا شوند. ما به این برنامه را اجرا می کنیم و با نرخ اشکال 10^{-4}، تسک‌ها را شکست خورده در نظر می‌گیریم و با استفاده از الگوریتم طراحی شده (GRIS) آن‌ها را بازیابی می‌کنیم. برای این شبیه سازی هر مجموعه شامل 10 تسک در نظر گرفته شد و بازه‌ی زمانی این تسک‌ها بین 0.1 تا 100 در نظر گرفته شده است. احتمال خرابی این سیستم به ازای الگوریتم GRIS و 10000 مجموعه تسک به صورت میانگین 10^{-4} بدست آمد.

C. پیاده سازی الگوریتم کاهش سطح فرکانسی

در این مقاله، جهت افزایش قابلیت اطمینان سیستم به معرفی الگوریتمی برای کاهش سطح فرکانسی اجرای تسک‌ها پرداخته شده است. اما همانطور که در این مقاله اشاره شده، از ذکر جزئیات شبیه سازی این الگوریتم خودداری شده است. با این حال ما به پیاده سازی

این الگوریتم می پردازیم و به صورت پیش فرض پارامترهای تعیین شده در این الگوریتم را مقدارهی می‌کنیم. جهت شبیه سازی این الگوریتم، در تسک‌های تولید شده یک پارامتر جدید به عنوان توان مصرفی هر تسک علاوه بر سایر پارامترها که پیش تر اشاره کردیم، در نظر می‌گیریم که از آن برای محاسبه نسبت انرژی مصرفی- زمان (رابطه‌ی (10)) استفاده می‌کنیم. همچنین به پیاده سازی تابعی برای محاسبه‌ی نسبت انرژی مصرفی- زمان می‌پردازیم.

6- نتیجه‌گیری

ما به معرفی یک چارچوب جهت بهبود سطح SER سیستم با توجه به حفظ LTR و محدودیت‌های سیستم‌های بلادرنگ پرداختیم. سطح SER با استفاده از زمانبدی ایستا و تخصیص بازیابی پویای تسک‌ها بهبود می‌یابد. همچنین سطح LTR سیستم نیز با کاهش سطح فرکانسی برای تسک‌هایی که انرژی زیادی مصرف می‌کنند و یا اولویت کمتری دارند، بهبود می‌یابد. نتایج شبیه سازی‌ها نشان می‌دهد که روش ما در بهبود سطح SER در مقایسه با روش‌های تخصیص بازیابی ایستای موجود و روش‌های تخصیص بازیابی بدون نقض محدودیت‌های سیستم‌های بلادرنگ و حفظ سطح LTR ، بهتر عمل می کند.

فهرست مراجع

[1] G. Macario, M. Torchiano, and M. Violante, “An in-vehicle infotainment software architecture based on Google Android,” in Proc. Int. Symp. Industrial Embedded Systems, Jul. 2009, pp. 257–260.

[2] B. Zhao, H. Aydin, and D. Zhu, “Enhanced reliability-aware power management through shared recovery technology,” in Proc. Int. Conf. Computer-Aided Design, Nov. 2009, pp. 63–70.

[3] L. Huang, F. Yuan, and Q. Xu, “Lifetime reliability-aware task allocation and scheduling on MPSoC platform,” in Proc. Design, Automation and Test in Europe, Mar. 2009, pp. 51–56.

[4] B. Zhao, H. Aydin, and D. Zhu, “Generalized reliability-oriented energy management for real-time embedded applications,” in Proc. Design, Automation Conf., Jun. 2011, pp. 381–386.

[5] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, “Enhancing multicore reliability through wear compensation in online assignment and scheduling,” in Proc. Design, Automation and Test in Europe, Mar. 2013, pp. 1373–1378..

[6] B. Zhao, H. Aydin, and D. Zhu, “Energy management under general task-level reliability constraints,” in Proc. Int. Conf. the Real-Time and Embedded Technology and Application Symp., Apr. 2012, pp. 285–294.

[7] B. Nahar and B. Meyer, “Rotr: Rotational redundant task mapping for fail-operational MPSoCs,” in Proc. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Oct. 2015, pp. 21–28.

[8] Z. Al-bayati, B. Meyer, and H. Zeng, “Fault-tolerant scheduling of multicore mixed-criticality systems under permanent failures,” in Proc. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Sep. 2016, pp. 57–62.

[9] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, “Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs,” in Proc. Design, Automation and Test in Europe, Mar. 2014, pp. 1–6.

[10] J. Zhou, X. S. Hu, Y. Ma, and T. Wei, “Balancing lifetime and softerror reliability to improve system availability,” in Proc. Asia and South Pacific Design Automation Conf., Jan. 2016, pp. 685–690.

[11] Q. Han, M. Fan, and G. Quan, “Energy minimization for fault tolerant real-time applications on multiprocessor platforms using checkpointing,” in Proc. Int. Symp. Low Power Electronics and Design, Aug. 2013, pp. 76–81.