



دانشگاه کاشان  
University of Kashan

مجله محاسبات نرم

## SOFT COMPUTING JOURNAL

تارنمای مجله: [scj.kashanu.ac.ir](http://scj.kashanu.ac.ir)



### مروری بر روش‌های کاهش سربار نقطه‌وارسی

مصطفی آغبلاغی تبریزی<sup>1\*</sup>، دانشجوی کارشناسی ارشد، آراین شهبازیان<sup>1</sup>، دانشجوی کارشناسی ارشد، کیمیا شاه‌بختی<sup>1</sup>، دانشجوی کارشناسی ارشد، کوشیار زمانی<sup>1</sup>، دانشجوی کارشناسی ارشد، نوید کنعانی<sup>1</sup>، دانشجوی کارشناسی ارشد، سید امیر اصغری<sup>2</sup>، دانشیار، محمدرضا بینش مروستی<sup>1</sup>، دانشیار

<sup>1</sup> گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران.

<sup>2</sup> گروه علوم و فناوری داده، دانشکده مهندسی سامانه‌های هوشمند، دانشگاه تهران، تهران، ایران.

#### چکیده

امروزه تحمل‌پذیری اشکال در سیستم‌های مختلف امری ضروری است. استفاده از روش‌های ایجاد نقطه‌وارسی و نقاط امن برای بازگشت به هنگام خرابی باعث افزایش قابلیت اطمینان سیستم‌ها می‌شود. چالش اصلی در استفاده از روش‌های نقطه‌وارسی سربار آنها است. این سربار بر اثر اجرای فرآیند ایجاد نقطه‌وارسی است و باعث کاهش عملکرد کلی سیستم‌ها می‌شود. روش‌های بسیاری تاکنون به حل این مشکل پرداخته‌اند. این روش‌ها تلاش می‌کنند تا سربار ناشی از نقطه‌وارسی کاهش یابد و در نتیجه سیستم به حداکثر کارایی برسد. در این مقاله به مطالعه و مرور روش‌های گوناگون پیرامون کاهش سربار نقطه‌وارسی پرداخته شده است. این روش‌ها در گروه‌های مختلفی دسته‌بندی شده‌اند. دسته‌بندی‌ها بر اساس نوع پیاده‌سازی نقطه‌وارسی و سطوح مختلف سیستم‌ها مشخص گردیده‌اند. این دسته‌بندی‌ها شامل: نقطه‌های واری هم‌هنگ شده، نقطه‌وارسی در سطح سیستم، نقطه‌وارسی در سطح برنامه و نقطه‌وارسی در سیستم‌های محاسباتی توزیع شده است. در پایان با نمایش کلی دسته‌بندی‌های گفته شده در یک جدول جامع برای هر دسته‌بندی نتیجه‌گیری می‌شود.

#### اطلاعات مقاله

تاریخچه مقاله:

دریافت 8 اردیبهشت ماه 1403

پذیرش 14 مرداد ماه 1403

کلمات کلیدی:

نقطه‌وارسی

کاهش سربار نقطه‌وارسی

نقطه‌وارسی هم‌هنگ شده

نقطه‌وارسی در سطح سیستم

نقطه‌وارسی در سطح برنامه

نقطه‌وارسی در سیستم‌های محاسباتی توزیع

شده

© 1403 نویسندگان. مقاله با دسترسی آزاد تحت مجوز CC-BY

### 1. مقدمه

نقطه‌وارسی، باعث افزایش کارایی سیستم می‌شود. نقطه‌وارسی با وجود تحقق اهدافی مانند کاهش اتلاف زمان در هنگام مواجهه با خرابی و شکست، سربار قابل توجهی بر سیستم‌ها دارد. این سربار می‌تواند با افزایش زمان اجرای کلی برنامه‌ها باعث کاهش کارایی نهایی سیستم شود [2].

زمانی که یک سیستم سعی در ایجاد نقطه‌وارسی می‌کند با سربارهایی روبه‌رو است. این سربار به دلیل توقف اجرای برنامه اصلی سیستم به هنگام ایجاد نقطه‌وارسی ایجاد می‌شود. این وقفه

نقطه‌وارسی<sup>1</sup> روشی پرکاربرد برای افزایش تحمل‌پذیری اشکال در سیستم‌های کامپیوتری است [1]. از سرگیری دوباره فعالیت سیستم پس از مواجهه با خطا با استفاده از اطلاعات ذخیره شده در

✧ نوع مقاله: مروری

\* نویسنده مسئول

پست(های) الکترونیک: [m.aghbolaghitaibri@khu.ac.ir](mailto:m.aghbolaghitaibri@khu.ac.ir) (آغبلاغی تبریزی)

[s.a.asghari@ut.ac.ir](mailto:s.a.asghari@ut.ac.ir) (اصغری)

<sup>1</sup> Checkpoint

نحوه ارجاع به مقاله: مصطفی آغبلاغی تبریزی، آراین شهبازیان، کیمیا شاه‌بختی، کوشیار زمانی، نوید کنعانی، سید امیر اصغری، محمدرضا بینش مروستی، «مروری بر روش‌های کاهش سربار نقطه‌وارسی»، مجله محاسبات نرم، جلد 14، شماره 1، ص 134-153، بهار و تابستان 1404.

کاهش تعداد نقطه‌واری زمان توقف سیستم برای انجام فرآیند نقطه‌واری کاهش می‌یابد. این بهبود باید به گونه‌ای باشد که باعث کاهش قابلیت اطمینان سیستم نشود. در سیستم‌های مختلف، فاکتورهای متفاوتی را می‌توان ملاک این ایده قرار داد و با بررسی آنها اقدام به ایجاد یا عدم ایجاد نقطه‌واری کرد. فرآیند ایجاد نقطه‌واری در پژوهش‌های چند سال اخیر مورد توجه بسیار قرار گرفته است. اینکه از چه اطلاعاتی برای ایجاد نسخه پشتیبان استفاده شود و یا چه بخش‌هایی در طول زمان اجرای فرآیند نقطه‌واری صرف کد فعالیتهای خود، از زمینه‌های مورد بررسی محققان بوده است.

به دلیل گسترده بودن کاربرد نقطه‌واری، کاهش سربار نقطه‌واری می‌تواند در سطوح مختلفی انجام شود. نقطه‌واری می‌تواند شامل حالت پردازنده باشد. حالت پردازنده مقادیر ثابت‌های فرار است که طی فرآیند نقطه‌واری داخل حافظه غیرفرار ذخیره می‌شوند. در اغلب روش‌های معرفی شده توسط محققان تلاش شده است که با کمترین سربار ممکن بتوان به بیشترین کارایی در سیستم دست یافت.

در گذشته مقاله‌های مروری مختلفی در ارتباط با نقطه‌واری و مفاهیم پایه‌ای آن منتشر شده است [8]. مبحث نقطه‌واری با تمام سادگی در مفهوم، در جزئیات بسیار پر محتوا و قابل بحث است. بررسی‌های مختلف در مفاهیم نقطه‌واری این نتیجه را داشته است که توجه بسیاری از مقالات، پیرامون کاهش سربار نقطه‌واری بوده است.

در این کار با استفاده از ویژگی‌ها و نتایج گزارش شده از هر روش و اتکا به ادعای نویسندگان اقدام به مقایسه و بررسی میان روش‌های کاهش سربار نقطه‌واری شده است. در این مقاله معیار مقایسه جدیدی معرفی نشده است و فقط به مرور ادبیات پرداخته شده است. روش‌های کاهش سربار نقطه‌واری در دسته‌بندی‌های مختلفی معرفی شده‌اند. هر روش به صورت کلی معرفی شده است و به بیان جزئیات و الگوریتم‌های دقیق هر روش پرداخته نشده است. پس از معرفی روش‌های هر دسته‌بندی، توسط جدول به بیان مزایا و معایب هر روش پرداخته شده است. این تحلیل نظری بوده و با استفاده از

زمانی خود باعث افزایش زمان کلی اجرا می‌شود و در نهایت منجر به کاهش کارایی کلی سیستم خواهد شد [3].

از دیگر دلایل ایجاد سربار نقطه‌واری عملیات نوشتن در حافظه پایدار است. یک حافظه پایدار مکانی برای ذخیره نقطه‌واری است تا در صورت بروز اشکال بتوان حالت سیستم را از این حافظه بازیابی کرد. نوشتن نقطه‌واری در حافظه پایدار باعث ایجاد شلوغی در ورودی‌های خواندن و نوشتن حافظه می‌شود. این شلوغی در نهایت سبب تاخیر دسترسی به حافظه پایدار می‌شود و همچنین دیگر برنامه‌ها نیز تحت تاثیر این تاخیر قرار می‌گیرند [4].

دلایل ذکر شده این نکته را به روشنی بیان می‌کنند که بایستی کوشید تا حد امکان از سربار نقطه‌واری کم کرد تا کارایی کلی سیستم کاهش نیابد. روش‌های بررسی شده در این مقاله اغلب به دنبال کاهش سربار نقطه‌واری با کنترل همین دلایل گفته شده هستند.

در برخی سیستم‌ها، فرآیند نقطه‌واری در سطح سیستم و پنهان از برنامه‌ها صورت می‌پذیرد و در برخی دیگر، در سطح برنامه‌ها و کاربر انجام می‌شود. پیاده‌سازی نقطه‌واری چه در سطح نرم‌افزار و چه در سطح سخت‌افزار باشد، نیاز به حافظه‌ای برای ذخیره نقطه‌واری دارد. انتخاب نوع حافظه، تعداد و نحوه اجرای فرآیند نقطه‌واری و اطلاعات آن، نکاتی هستند که در روش‌های بررسی شده در این مقاله به آنها پرداخته شده است. برخی از نقطه‌های واری در حافظه‌های غیرفرار ذخیره می‌شوند تا حتی با قطعی برق هم به هنگام فراهم بودن شرایط قابل دسترسی باشد. همچنین نقطه‌های واری مبتنی بر دیسک سخت به دلیل مشکل پهنای باند، با ضعف کارایی همراه هستند [5]. در دهه گذشته مطالعات بسیاری برای کاهش هزینه‌های نقطه‌واری با استفاده از فناوری‌های نوظهور حافظه‌های غیرفرار صورت گرفته است [6]. چالش دیگر، انتخاب درست زمان ایجاد نقطه‌واری است که در روش‌های مختلف فاکتورهای گوناگونی بسته به نوع سیستم در نظر گرفته شده است [7].

برخی روش‌ها با کاهش تعداد نقطه‌واری با توجه به شرایط سیستم، در تلاش‌اند کارایی کلی سیستم را افزایش دهند. با

حالت سیستم سازگار باقی بماند. در نقطه‌وارسی هماهنگ شده، فرآیندها با همگام‌سازی فرآیند ایجاد نقطه‌های وارسی، یک وضعیت سازگار و بدون اثر سلسله‌مراتبی را در کل سیستم تضمین می‌کنند [11]. روش‌های مربوط به این دسته‌بندی در امر ایجاد یک نقطه‌وارسی سازگار و بدون اثر سلسله‌مراتبی با یکدیگر اشتراک دارند. در شکل (2) روش‌های مربوط به این دسته‌بندی نشان داده شده است.



شکل (2): روش‌های کاهش سربار نقطه‌وارسی هماهنگ شده

### 1.1.2. نقطه‌وارسی چند سطحی بدون دیسک<sup>3</sup>

با افزایش نرخ خرابی و شکست، سیستم‌های ایجاد نقطه‌وارسی بر پایه دیسک بایستی پهنای باند خود را افزایش دهند تا از بروز گلوگاه جلوگیری کنند [12]. اگر پهنای باند کافی در دسترس باشد، این روش کارایی بالایی ارائه می‌دهد. با وجود محدودیت پهنای باند دیسک، روش بدون دیسک گزینه خوبی برای ایجاد نقطه‌وارسی است؛ زیرا از ذخیره‌سازی پایدار<sup>4</sup> استفاده نمی‌کند و مشکل گلوگاه و تاخیر در پاسخ‌دهی ندارد [13].

گزارش‌های نویسندگان و ادعای آنها بیان شده است. معتز الزهی و همکاران [9]، در مروری کلی روش‌های مبتنی بر بازگشت - بازیابی را معرفی کرده‌اند. آنها در مطالعه‌های خود نگاهی کلی به سربار ناشی از نقطه‌وارسی داشته‌اند.

## 2. روش‌های کاهش سربار نقطه‌وارسی

در این بخش به بررسی روش‌های مختلف کاهش سربار نقطه‌وارسی که اخیراً معرفی شده‌اند، می‌پردازیم. این روش‌ها از نظر نوع نقطه‌وارسی به چندین دسته‌بندی تقسیم شده‌اند. این روش‌ها که در شکل (1) نمایش داده شده‌اند، در دسته‌بندی‌های نقطه‌وارسی هماهنگ شده، نقطه‌وارسی در سطح سیستم و حافظه‌ها، نقطه‌وارسی در سطح برنامه و محیط‌های محاسباتی توزیع‌شده و موازی گردآوری شده‌اند. این دسته‌بندی‌ها با بررسی و مطالعه روش‌های مختلف صورت گرفته است و سعی شده تا روش‌هایی با ویژگی‌های مشابه در یک دسته‌بندی قرار بگیرند.

در ابتدای هر بخش، دسته‌بندی مربوط معرفی شده و سپس در ادامه روش‌های مرتبط به آن دسته‌بندی بررسی شده است و در بخش 3، معیارهای بررسی شرح داده شده است.

### 1.2. نقطه‌وارسی هماهنگ شده<sup>1</sup>

در نقطه‌های وارسی هماهنگ شده نیاز است تا همه فرآیندها به منظور تشکیل یک حالت کلی سازگار، اقدام به تنظیم فرآیندهای نقطه‌وارسی خود کنند. در این‌گونه نقطه‌های وارسی، هر فرآیند تنها می‌تواند یک نقطه‌وارسی دائمی از خود ذخیره کند. این امر باعث کاهش سربار ذخیره‌سازی می‌شود و از اثر سلسله‌مراتبی<sup>2</sup> نیز جلوگیری می‌کند [10]. اثر سلسله‌مراتبی زمانی اتفاق می‌افتد که سیستم سعی دارد با بازگشت به نقطه‌های وارسی خود، یک حالت کلی سازگار ایجاد کند. این امر باعث می‌شود تا فرآیندهای دیگر سیستم نیز به دلیل وابستگی‌های موجود میان خود و فرآیند بازگشت داده شده، اقدام به بازگشت نمایند تا

<sup>3</sup> Multilevel Diskless Checkpointing

<sup>4</sup> Stable Storage

<sup>1</sup> Coordinated Checkpointing

<sup>2</sup> Domino effect



شکل (1): روش‌های کاهش سربار نقطه‌واری

می‌شود. در بخش اول با فرض نقطه‌وارسی آفلاین و دانستن طول کلی الگو پیش از شروع اجرا، اقدام به اثبات رابطه زمان مورد نیاز برای اجرای الگو می‌کنند. با توجه به فرمول اثبات شده در می‌یابند، زمانی که همه تکه‌ها در یک الگو دارای اندازه یکسانی باشند، زمان اجرا کلی مورد انتظار به حداقل می‌رسد. این مقدار یکسان همان نسبت اندازه کل الگو به تعداد تکه‌ها است. این قضیه در این مقاله اثبات شده است.

در قسمت بعدی نویسندگان به بررسی شرایط آنلاین و بدون داشتن طول کلی الگو پرداخته‌اند. آنها با داشتن تعداد تکه‌ها، الگوی بهینه دارای تکه‌ای هم اندازه را که نحوه محاسبه آن در زمان اجرا نشان داده شده است، بررسی کرده‌اند. در نهایت به این نتیجه دست می‌یابند که راه‌حل بهینه نقطه‌وارسی دوسطحی بر پایه زمان‌بندی‌های آنلاین و آفلاین است و راه‌حل بهینه باید فواصل نقطه‌وارسی با اندازه برابر را در سطوح نقطه‌وارسی خاص اتخاذ کند. نتایج به دست آمده از آزمایش این روش آنلاین پیشنهاد شده نشان می‌دهد که بهبودها مانند روش آفلاین است. با این روش، زمان واقعی از شروع تا پایان برنامه را می‌توان تا 25/3% در مقایسه با نتایج سایر روش‌های پیشرفته کاهش داد.

### 3.1.2. روش مبتنی بر گراف در سیستم‌های موبایل

از دیگر زمینه‌های استفاده از نقطه‌وارسی، سیستم‌های عامل سیار<sup>7</sup> است. یک عامل سیار، برنامه نرم‌افزاری است که می‌تواند به همراه داده‌هایی از یک کامپیوتر به کامپیوتر دیگری انتقال یابد. این سیستم‌ها هم با خرابی‌هایی نظیر شکست به هنگام درخواست انتقال، استثناهای ارتباطی، خرابی سیستم و یا نقض امنیت همراه هستند [18]. روش گراف تقدم در این گونه سیستم‌ها برای افزایش قابلیت اطمینان پیشنهاد شده است [19]. در این روش عامل‌های وابسته به هر عامل سیار به‌عنوان گره در گراف تقدم ذخیره می‌شوند. زمانی که عمق این گراف از یک اندازه مشخص بیشتر شود، عامل سیار درخواست نقطه‌وارسی می‌کند. این روش از نوع نقطه‌وارسی هماهنگ شده است. با

روش ارائه شده توسط Hakkariinen و همکاران که نقطه‌وارسی چندسطحی بدون دیسک نام دارد، کاهش سربرار بالای نقطه‌وارسی را تضمین می‌کند. این روش می‌تواند از یک، دو تو یا تعداد بیشتری خرابی هم‌زمان را تحمل کند و با استفاده از نقطه‌های واری عملیات بازیابی را انجام دهد. مراحل کلی اجرای این الگوریتم به این صورت است که ابتدا محاسبه می‌شود چه سطحی از نقطه‌وارسی پیاده‌سازی شود. سپس با هماهنگ کردن این نقطه‌های واری سعی در رسیدن به حالت سازگار دارد. برای شروع عملیات بازیابی پس از شناسایی خرابی، الگوریتم ابتدا سعی می‌کند از جدیدترین نقطه‌وارسی برای بازیابی وضعیت پردازنده ناموفق استفاده کند. در صورت بروز خرابی‌های هم‌زمان دیگر به هنگام بازیابی و عدم موفقیت در بازیابی، سیستم به آخرین حالت مناسب برای مدیریت کردن این تعداد خطا باز می‌گردد. طبق بررسی نویسندگان، این روش در مقایسه با روش‌های یک‌سطحی قبلی کاهش زیادی در زمان اجرا داشته است. دلیل این بهبود، نبود دسترسی آهسته به دیسک در این روش است که تاخیر ناشی از پهنای باند را از بین می‌برد و می‌تواند کارایی بالاتری ارائه کند.

### 2.1.2. روش نقطه‌وارسی آنلاین دوسطحی بهبودیافته

در سیستم‌های محاسباتی با کارایی بالا<sup>5</sup> در صورت بروز هر نوع خرابی اگر سیستم راه‌اندازی دوباره شود و محاسبات قبلی از دست برود، علاوه بر اتلاف منابع با کاهش قابل توجه کارایی مواجه هستیم. روش نقطه‌وارسی چندسطحی<sup>6</sup> برای رفع مشکل سربرار نقطه‌وارسی معرفی شده است [14] - [16].

در روش نقطه‌وارسی دوسطحی در سطح اول با خطاهای گذرا که سربرار نقطه‌وارسی کمی دارند (مانند خطاهای حافظه) سروکار داریم. سطح دوم بر خرابی‌های سخت‌افزاری مانند خرابی نودها یا گره‌ها تمرکز دارد. در این مقاله [17]، نویسندگان در ابتدا با فرموله کردن شرایط آنلاین و آفلاین به نتایجی دست می‌یابند که منجر به رسیدن یک روش بهینه برای نقطه‌وارسی چندسطحی

<sup>5</sup> High-Performance Computing (HPC)

<sup>6</sup> Multilevel Checkpointing

<sup>7</sup> Mobile Agent

میان دو فاکتور مصرف انرژی و قابلیت اطمینان یک چالش است. ایده فراهم کردن این دو با استفاده از یک روش نقطه‌وارسی، هدف نویسندگان این مقاله [21] است. تمرکز آنها معرفی یک روش نقطه‌وارسی تطبیقی است که بتواند کار هدررفته به دلیل خرابی را در سیستم کاهش دهد و زمان‌بندی بهینه‌ای را برای نقطه‌وارسی معرفی کند.

روش پیشنهادی شامل میانگین متحرک ساده، میانگین متحرک وزنی، میانگین متحرک نمایی، همبستگی خودکار شکست و کاهش نرخ خطر است. این روش‌ها به صورت پویا فواصل نقطه‌وارسی را بر اساس الگوهای خرابی تنظیم می‌کنند تا مصرف انرژی را بهینه کنند. این روش با در نظر گرفتن توزیع آماری زمان بین خرابی‌ها، یک برنامه‌ریزی برای نقطه‌وارسی اعلام می‌کند. محققان همچنین ذکر کرده‌اند که روش‌های بهینه‌سازی و کاهش سربار نقطه‌وارسی که در زمان اجرا، برنامه‌ریزی ایجاد نقطه‌وارسی انجام می‌دهند، برای اهداف مصرف انرژی بهینه نیستند. در واقع شاهد یک معاوضه بین صرفه‌جویی در انرژی و هزینه‌های سربار زمان اجرا به دلیل سربار نقطه‌وارسی هستیم. بررسی‌ها نشان می‌دهد روش پیشنهادی، با حداقل سربار کارایی می‌تواند تا 50% کاهش مصرف انرژی را نسبت به روش‌های پایه‌ای فراهم کند.

#### 6.1.2. انتخاب بهینه نقطه‌وارسی با افزونگی دو مدولار

امروزه با افزایش پیچیدگی سیستم‌های روی تراشه نرخ خرابی افزایش زیادی پیدا کرده است. از این رو بایستی روش‌هایی را برای تضمین قابلیت اطمینان این‌گونه سیستم‌ها بررسی کرد. تحمل اشکالات گذرا با انتخاب موقعیت‌های بهینه نقطه‌وارسی به منظور به حداقل رساندن سربار هدف پژوهش Kang و همکاران بوده است [22]. آنها با استفاده از افزونگی دو مدولار<sup>10</sup> توانسته‌اند یک زمان‌بندی برای نقطه‌وارسی پیشنهاد دهند که دارای سربار کمی است. آنها مدل خود را برای سیستم‌های تک پردازنده و چندپردازنده معرفی کرده‌اند. بر اساس الگوریتم، یک زمان‌بندی بهینه برای ایجاد نقطه‌وارسی

شروع درخواست نقطه‌وارسی، همه عامل‌های وابسته توسط عامل بررسی‌کننده از این عملیات باخبر می‌شوند. این عامل‌های بررسی‌کننده ارسال شده، توسط عامل اصلی درخواست‌دهنده وزن‌دهی می‌شوند. اگر عامل‌های وابسته با درخواست نقطه‌وارسی موافقت کنند، وزن عددی تاییدکننده را باز می‌گردانند. در صورت موافقت همه عامل‌ها با درخواست ارسال شده، نقطه‌وارسی موقتی، نهایی و ذخیره می‌شود. بعد از ذخیره نقطه‌وارسی گراف‌های تقدم قبلی پاک می‌شوند تا اندازه اطلاعات منتقل شده کاهش یابد. این روش تاخیر زمانی روش واریسی جهانی را به میزان قابل توجهی کاهش می‌دهد. از آنجا که در این روش تعداد کمی از عامل‌ها مجاز به ایجاد نقطه‌وارسی نهایی می‌شوند، سربار کمتری را نسبت به روش‌های دیگر از این نوع سیستم‌ها شاهد هستیم.

#### 4.1.2. استفاده از روش فعال نقطه‌وارسی

یکی دیگر از سیستم‌هایی که نیاز به پیاده‌سازی نقطه‌وارسی برای افزایش قابلیت اطمینان و تحمل‌پذیری اشکال دارد، پردازش ابری است. الگوریتم هماهنگ شده فعال تحمل‌پذیر اشکال<sup>8</sup> روشی است که اخیراً پیشنهاد شده است [20]. این روش با در نظر گرفتن ویژگی دمای پردازنده سعی می‌کند تا ماژول‌های نزدیک به خرابی را پیش‌بینی کند. بعد از پیش‌بینی ماژول نزدیک به خرابی آن را با ماژول‌های بهتر جایگزین می‌کند. این امر با انتخاب بهترین ماژول از بین لیستی از ماژول‌ها با الگوریتم PSO<sup>9</sup> انجام می‌شود. با این کار علاوه بر بهبود زمان کلی اجرا، از خرابی‌هایی که منجر به عملیات بازیابی می‌گردند تا حد زیادی جلوگیری می‌شود. می‌توان گفت با استفاده از ویژگی ذکر شده از سربار بازیابی و نقطه‌وارسی کاست.

#### 5.1.2. ایجاد یا عدم ایجاد نقطه‌وارسی، تقابل مصرف انرژی و سربار

در سیستم‌های محاسباتی با کارایی بالا با مقیاس بزرگ، تعادل

<sup>8</sup> Proactive Coordinated Fault-Tolerance (PCFT)

<sup>9</sup> Particle Swarm Optimization (PSO)

<sup>10</sup> Dual Modular Redundancy (DMR)

هزینه کنترلی متغیرها برای ایجاد نقطه‌وارسی کاهش یابد. این روش که بهبودی بر روش‌های پیشین است توانسته سربار نقطه‌وارسی و راه‌اندازی دوباره را تا 60% نسبت به روش‌های سنتی کاهش دهد. این روش علی‌رغم بهبود در سربار نقطه‌وارسی باعث افزایش استفاده از حافظه پردازنده می‌شود که نیازمند بررسی است.

### 8.1.2. SCR کتابخانه‌ای مقیاس‌پذیر برای نقطه‌وارسی

در سیستم‌های محاسباتی با کارایی بالا که شاهد پیشرفت قابل توجهی در قطعات سخت‌افزاری آنها هستیم، ایجاد یک نقطه‌وارسی بهینه امری ضروری است. در سیستم‌هایی با مقیاس بزرگ یک فرآیند نقطه‌وارسی ممکن است تا ده‌ها دقیقه زمان ببرد [25]. روش نقطه‌وارسی چندسطحی مشکل هزینه بالای نقطه‌وارسی را حل می‌کند [26].

برای پیاده‌سازی آسان نقطه‌وارسی چندسطحی در سیستم‌های مقیاس بزرگ کتابخانه SCR توسط Moody و همکاران پیشنهاد شده است [27]. این کتابخانه نقطه‌وارسی را علاوه بر نوشتن روی سیستم فایل موازی، در RAM، FLASH و یا دیسک ذخیره می‌کند.

ایده اصلی SCR از دو مشاهده کلیدی الهام گرفته شده است. اول این که هر فرآیند تنها به آخرین نقطه‌وارسی خود نیاز دارد؛ دوم اینکه یک خرابی گذرا تنها قسمت کوچکی از سیستم را از کار می‌اندازد. SCR تنها کش آخرین نقطه‌وارسی را ذخیره می‌کند و نقطه‌های واریسی جدید جایگزین موارد گذشته می‌شوند. با رخداد خرابی SCR تلاش می‌کند که آخرین نقطه‌وارسی را از کش بازیابی کند؛ اگر در این کار با شکست مواجه شود، پس از دریافت آخرین نقطه‌وارسی از سیستم فایل موازی، فرآیند راه‌اندازی مجدد می‌شود. این روش به دلیل کاهش قابل توجه سربار نقطه‌وارسی می‌تواند تا 1000 برابر به هنگام ذخیره محلی سریع باشد. طبق آزمایش‌های انجام شده با این روش در 96/4% خرابی‌ها می‌توان با استفاده از نقطه‌های واریسی ذخیره شده در کش بازیابی را انجام داد.

در مرز اجرای وظیفه<sup>1</sup> محاسبه می‌شود. با استفاده از معرفی فاکتور بودجه زمانی<sup>2</sup> سعی در یافتن نقطه‌وارسی بهینه می‌کند تا تا این آستانه نقض نشود. در این روش که سعی در بهبود کارهای قبلی با تمرکز بر کاهش سربار نقطه‌وارسی شده است، طبق نتایج مقایسه‌های صورت گرفته روش پیشنهادی به طور متوسط 68/6 برابر برای تک پردازنده‌ها و 86/2 برابر برای چندپردازنده‌ها کاهش سربار نقطه‌وارسی را به همراه داشته است.

### 7.1.2 HeteroCheckpoint نقطه‌وارسی کارآمد برای

#### شتاب‌دهنده‌ها

افزایش استفاده از واحدهای پردازش گرافیکی<sup>3</sup> در سیستم‌های محاسباتی با کارایی بالا منجر به نرخ خرابی بالاتر در مقایسه با سیستم‌هایی می‌شود که از پردازنده‌های عادی استفاده می‌کنند. افزایش تحمل‌پذیری اشکال در این سیستم‌ها نیاز به ایجاد نقطه‌وارسی است.

روش‌های سنتی نقطه‌وارسی به دلیل ایجاد ترافیک داده‌ای بالا بین واحد پردازش گرافیکی و حافظه ناکارآمد هستند [23]. این مشکل را می‌توان با کاهش سربار نقطه‌وارسی برطرف کرد. روش یکپارچه نقطه‌وارسی در CPU-GPU به حل مشکل ترافیکی ایجاد شده به دلیل سربار نقطه‌وارسی پرداخته است [24].

پیش‌کپی داده‌ها در سطح تکه<sup>4</sup> می‌تواند به طور موثر زمان اجرای اجرای کلی برنامه را کاهش دهد. این عملیات پیش‌کپی قبل از شروع عملیات نقطه‌وارسی هماهنگ انجام می‌شود. همچنین می‌توان تکه‌های بدون تغییر را شناسایی کرد و از کپی‌های غیرضروری از حافظه غیرفرار به حافظه گرافیکی پویا جلوگیری نمود. این کار باعث کاهش قابل توجهی در سربار نقطه‌وارسی می‌شود. روش دیگر استفاده از نشان‌گذاری علامت کثیف برای متغیرهای تغییر یافته در زمان اجرا است. این نشان‌گذاری پویا توسط واحد پردازش گرافیکی انجام می‌شود و باعث می‌شود

<sup>1</sup> Task Execution Boundary

<sup>2</sup> Time Budget

<sup>3</sup> GPU

<sup>4</sup> Chunk

درخواست‌های خواندن از حافظه شود. زمانی که می‌خواهیم از حافظه فرار نقطه‌وارسی بگیریم و در یک حافظه غیرفرار ذخیره کنیم [30]، بایستی اطلاعات را از حافظه بخوانیم و منتقل کنیم. حال اگر درخواست خواندن از سمت پردازنده به حافظه برسد، تداخل با درخواست نقطه‌وارسی نتیجه کاهش عملکرد را می‌دهد [31].

نویسندگان مقاله [32]، روشی با نام Shadow پیشنهاد داده‌اند که با استفاده از مولفه‌های سخت‌افزاری و نرم‌افزاری سعی در پیش‌بینی دوره‌های زمانی بیکار بانک‌های حافظه دارد تا بتواند از این زمان‌ها برای انجام عملیات نقطه‌وارسی استفاده کند. همچنین با این کار تداخل درخواست نقطه‌وارسی با درخواست‌های برنامه‌ها را به حداقل برساند. در صورتی که با وجود پیش‌بینی، تداخلی میان درخواست‌های نقطه‌وارسی و برنامه صورت گیرد، Shadow اولویت درخواست‌های برنامه‌ها را افزایش می‌دهد تا عملکرد سیستم کاهش نیابد. روش پیشنهادی نویسندگان بر مبنای روش‌های Stop-And-Copy و Pre-Copy است که با پیگیری صفحات حافظه تنها اقدام به ایجاد نقطه‌وارسی از صفحات تغییر یافته می‌کند. روش Shadow هزینه‌های نقطه‌وارسی را نسبت به دیگر روش‌های بر پایه Pre-copy و Stop-And-Copy به ترتیب 42% و 16% در میانگین کاهش می‌دهد.

### 2.2.2. تصویر لحظه‌ای درخواستی، یک سیستم نسخه‌سازی کارآمد

یکی دیگر از کاربردهای نقطه‌وارسی استفاده در حافظه‌های فاز متغیر<sup>1</sup> است. در روش تصویر لحظه‌ای درخواستی<sup>2</sup> نویسندگان با کاهش سربار نوشتن و به حداقل رساندن نوشتن‌های بیهوده به هنگام تهیه تصویر لحظه‌ای، سعی در افزایش کارایی داشتند [33]. آنها در روش خود به جای نوشتن تمام فایل، تنها بخش‌های تغییر یافته را ذخیره کرده‌اند. در این روش ابتدا لیستی از تصاویر لحظه‌ای تهیه می‌شود که هر نسخه از این لیست شامل تصاویر لحظه‌ای وابسته قبلی است و اطلاعات بلاک‌هایی است که

### 2.2. روش‌های نقطه‌وارسی در سطح سیستم و حافظه‌ها

در این بخش به بررسی روش‌های کاهش سربار نقطه‌وارسی در حافظه‌ها و حالت اصلی سیستم می‌پردازیم. در این روش‌ها سعی شده تا با کمترین سربار بتوان بیشترین قابلیت اطمینان را برای حافظه‌ها در سیستم‌های مختلف فراهم کرد.

نقطه‌وارسی در سطح حافظه به معنی ایجاد یک تصویر لحظه‌ای از حالت کل برنامه موجود در حافظه است. به هنگام اشکال، این حالت ذخیره شده در حافظه خارجی جهت ادامه فعالیت سیستم از یک نقطه امن بازخوانی می‌شود [28]. نقطه‌وارسی در سطح سیستم‌ها سعی دارد تا به دور از نگاه نرم‌افزار اقدام به ذخیره حالت پردازش درون پردازنده کند و تحمل‌پذیری اشکال را به ارمغان بیاورد [29]. روش‌های ذکر شده در این دسته‌بندی از نظر پنهان بودن عملیات از دید نرم‌افزار با یکدیگر اشتراک دارند. همچنین دیگر اشتراک آنها ایجاد نقطه‌وارسی از حالت درونی حافظه‌ها یا پردازنده است. در شکل (3) روش‌های مربوط به این دسته‌بندی نمایش داده شده است.



شکل (3): روش‌های کاهش سربار نقطه‌وارسی در سطح سیستم و حافظه‌ها

### 1.2.2. بهبود موازی‌سازی سطح بانک برای نقطه‌وارسی در حافظه

نقطه‌وارسی در سطح بانک‌های حافظه می‌تواند باعث تداخل در

<sup>1</sup> Phase-Change Memory (PCM)

<sup>2</sup> On-Demand Snapshot

حافظه را نسبت به روش‌های گذشته به ارمغان می‌آورد. فرض آزمایشی استفاده شده در مقایسه این روش با روش‌های گذشته وجود کمتر از 100 دستور Add/Remove بین دو نقطه‌وارسی است. علاوه بر این، طرح‌های DSIC-undo و DSIC-redo می‌توانند بین یکدیگر جابجا شوند و پیچیدگی محاسباتی را برای برنامه‌های متمرکز بر نوشتن و خواندن کاهش دهند.

#### 4.2.2. نقطه‌وارسی سبک در حافظه‌های مجازی موبایل

در سیستم‌هایی که با مجازی‌سازی اقدام به اجرای چند سیستم‌عامل کرده‌اند، ایجاد نقطه‌وارسی با هزینه‌های بالایی همراه است. این هزینه علاوه بر زمان، از نظر میزان حافظه موردنیاز برای ذخیره‌سازی نیز مشکل‌ساز است. داده‌های ذخیره‌شده برای نقطه‌وارسی در این‌گونه عامل‌های سیار می‌تواند تا حد چند گیگابایت هم‌افزایش یابد [38]. برای پیاده‌سازی یک رویکرد سبک نقطه‌وارسی، روش نقطه‌وارسی سبک و سریع<sup>2</sup> معرفی شده است [39].

در این روش صفحات حافظه مجازی دسته‌بندی می‌شوند. این دسته‌بندی‌ها شامل حالات ماشین مجازی، مجموعه صفحات در حال کار و مجموعه صفحات بیکار هستند. این روش صفحات در حال کار را در حافظه غیرفرار ذخیره می‌کند که سرعت بالاتری نسبت به حافظه فلش دارد. صفحات بیکار در حافظه فلش ذخیره می‌شوند. مجموعه صفحات در حال کار باید تا حد امکان شامل صفحاتی باشد که پس از بازیابی مورد استفاده ماشین مجازی قرار می‌گیرد. رویکرد اصلی نویسندگان تفاوت میان اطلاعات نقطه‌وارسی و همچنین فضای ذخیره‌سازی آنها بوده است. با این کار آنها ترافیک خواندن و نوشتن حافظه را مدیریت کرده و کاهش داده‌اند. با ایجاد نقطه‌وارسی از صفحات در حال کار، سربار ناشی از نقطه‌وارسی را کاهش داده‌اند. این روش در مقایسه با روش‌های نقطه‌وارسی ساده از ماشین‌های مجازی، باعث کاهش 34% در فعالیت‌های نوشتن می‌شود. همچنین به میزان 50% بهبود در مصرف انرژی دارد.

بازیابی شده‌اند. روش تصویر لحظه‌ای درخواستی بلاک‌های خالی را برای نوشتن داده نقطه‌وارسی اختصاص می‌دهد. برای وضعیت‌هایی که چیزی برای انتقال به لیست تصویر لحظه‌ای وجود ندارد، از اشاره‌گر تهی برای حفظ وضعیت قبل از ایجاد بلوک استفاده می‌کند. در عملیات بازیابی هر بلوک با توجه به لیست تصویر لحظه‌ای به حالت قبلی بازگردانده می‌شود. این کار با به‌روزرسانی یا تغییر اشاره‌گرهای مربوط به لیست انجام می‌شود. روش پیشنهادی این به‌روزرسانی‌ها را به صورت بازگشتی پیاده‌سازی می‌کند. این روش در مقایسه با روش ZFS [34]، که یک سیستم نسخه‌سازی فایل است، تا 99% سربار نوشتن نقطه‌وارسی را کاهش می‌دهد. علاوه بر این به دلیل نوشتن کمتر مصرف انرژی کمتری دارد.

#### 3.2.2. نقطه‌وارسی افزایشی، یک رویکرد بر مبنای ساختار

داده

با افزایش تقاضا برای پردازش با سرعت بالا، پایگاه‌های داده داخل حافظه به طور گسترده‌ای برای تجزیه و تحلیل مورد استفاده قرار می‌گیرند. روش نقطه‌وارسی افزایشی مبتنی بر ساختار داده کار گروه تحقیقاتی WANG است [35]. روش‌هایی مانند ARIES Logging و Command Logging وجود دارند که دارای سربار زیادی هستند [36] و [37]. روش نقطه‌وارسی افزایشی بر مبنای ساختار داده<sup>1</sup> با استفاده از ویژگی‌های نامربوط مجموعه‌ها اقدام به بررسی اعتبارات دستورات و حذف دستورات اضافی می‌کند. این امر کاهش بسیار زیاد در هزینه ذخیره‌سازی و زمان ایجاد لاگ را فراهم می‌کند. نویسندگان برای تحقق این امر دو طرح DSIC-undo و DSIC-redo طراحی کرده‌اند. از این دو طرح برای بازیابی آخرین وضعیت در زمان واقعی و برای پخش مجدد دستورات استفاده می‌شود. این طرح‌ها دارای مجموعه دستوراتی هستند که هر یک الگوریتمی را پیاده‌سازی می‌کنند.

روش DSIC با حفظ ویژگی‌های مثبت روش‌های پیشین، نزدیک به 30% بهبود زمان ایجاد نقطه‌وارسی و 40% کاهش فضای

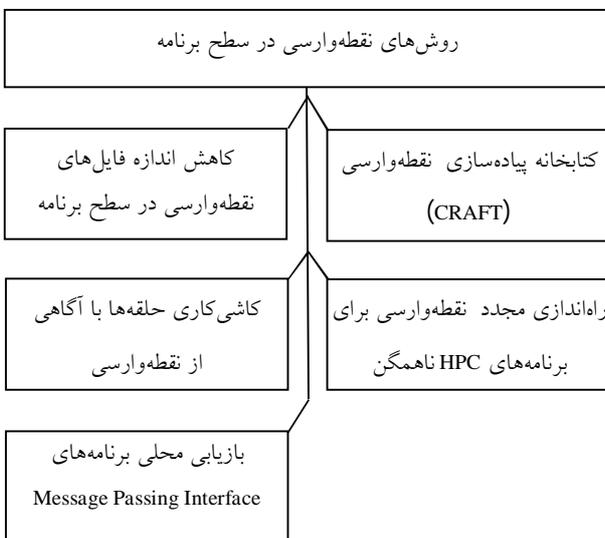
<sup>2</sup> Fast and Lightweight Checkpointing (FLIC)

<sup>1</sup> Data Structure based Incremental Checkpointing (DSIC)

### 1.3.2. کاهش اندازه فایل‌های نقطه‌وارسی در سطح برنامه

در سیستم‌های محاسباتی با کارایی بالا چالش دیگر ایجاد نقطه‌وارسی در برنامه‌های موازی با زمان اجرای طولانی است. در این گونه برنامه‌ها روش‌های سنتی نقطه‌وارسی باعث ایجاد فایل‌های حجیمی می‌شوند [42].

هدف نویسندگان مقاله [43]، ارائه روشی برای کاهش اندازه فایل نقطه‌وارسی در سطح برنامه است. با این کار سربار ناشی از نقطه‌وارسی به طور قابل توجهی کاهش می‌یابد. در روش پیشنهادی بر تجزیه و تحلیل متغیرهای زنده برنامه‌ها تاکید شده است. متغیرهای زنده شامل متغیرهایی می‌شوند که به هنگام اجرای برنامه از آنها استفاده می‌شود. این روش در نظر دارد تا در زمان ایجاد فایل‌های نقطه‌وارسی تنها این گونه متغیرها را ذخیره کند. نویسندگان به بررسی روش‌های نقطه‌وارسی افزایشی و روش‌های فشرده‌سازی فایل‌های نقطه‌وارسی پرداخته‌اند. استفاده از فشرده‌سازی باعث افزایش 2% کارایی نقطه‌وارسی نسبت به روش‌های بدون فشرده‌سازی می‌شود.



شکل (4). روش‌های کاهش سربار نقطه‌وارسی در سطح برنامه

### 2.3.2. راه‌اندازی مجدد نقطه‌وارسی برای برنامه‌های HPC ناهمگن

برای راحتی پیاده‌سازی یک روش بهینه نقطه‌وارسی در سطح برنامه‌ها، کتابخانه FTI معرفی شده است [44]. این کتابخانه

### 5.2.2. Survive. روش نقطه‌وارسی افزایشی بر مبنای اشاره‌گر

در مقاله [40]، نویسندگان روشی بهبود یافته برای نقطه‌وارسی در سطح حافظه‌های پویا معرفی کرده‌اند. اساس این روش بر مبنای نقطه‌وارسی افزایشی است [41]. در مواقع قطعی برق به دلیل فرار بودن اطلاعات ذخیره شده نقطه‌وارسی امکان از سرگیری دوباره فعالیت‌ها وجود ندارد. نویسندگان در روش خود که آن را Survive نامیده‌اند، با استفاده از ذخیره آخرین حالت در یک حافظه غیر فرار این امکان را فراهم کرده‌اند که در صورت قطعی برق هم بتوان دوباره اجرا را با وصل شدن برق از سر گرفت. در این روش از یک لیست نقطه‌وارسی برای پیاده‌سازی نقطه‌وارسی افزایشی استفاده می‌شود. علاوه بر این از اشاره‌گر برای هر ردیف این لیست استفاده می‌شود تا آخرین نسخه هر یک از ردیف‌ها را در پایان هر دوره پیگیری کنند. برای افزودن این اشاره‌گر به ردیف‌های لیست از بیت‌های فراداده استفاده می‌شود. در پایان هر دوره با استفاده از این اشاره‌گرها آخرین نسخه هر ردیف داخل حافظه غیر فرار ذخیره می‌گردد. حافظه غیر فرار در صورت مشغول نبودن حافظه توسط دستورات خواندن و نوشتن، به روزرسانی می‌شود. با منتقل شدن ردیف‌های لیست نقطه‌وارسی به حافظه غیر فرار می‌توان ورودی‌ها را از لیست پاک کرد. این روش نسبت به روش غیر بهینه تنها 3/5% سربار زمان اجرا تحمیل می‌کند اما در مقابل، سازگاری در برابر خرابی را فراهم می‌کند.

### 3.2. روش‌های نقطه‌وارسی در سطح برنامه

روش‌های ایجاد نقطه‌وارسی علاوه بر سطح سیستم، در سطح برنامه هم دارای کاربردهای فراوانی است. محققان تلاش کرده‌اند تا در این زمینه روش‌ها و کتابخانه‌هایی را پیشنهاد دهند تا بتوان به آسانی فرآیند نقطه‌وارسی را در برنامه‌ها پیاده‌سازی کرد. همچنین روش‌هایی که با تغییر در کد برنامه‌ها سعی در ایجاد یک روش نقطه‌وارسی با سربار کم دارند، در این دسته قرار گرفته‌اند. روش‌های مربوط به این دسته‌بندی در شکل (4) نشان داده شده است.

در روش کاشی‌کاری<sup>1</sup> با تغییر کد حلقه به حالتی که اجرا به صورت کاشی‌طور باشد از وابستگی داده‌ای چند نود در هر کاشی جلوگیری می‌شود که در نتیجه نیازی به ایجاد نقطه‌وارسی از آنها نیست. نویسندگان علاوه بر معرفی الگوریتم برای حلقه‌های دوسطحی، برای حلقه‌های سه سطحی و چندسطحی هم پژوهش خود را تکمیل کرده‌اند و نتایج تجربی از نظر میزان داده هر نقطه‌وارسی و زمان کلی اجرا و ترمیم را عنوان کرده‌اند. این روش در مقایسه با روش عادی نقطه‌وارسی از حلقه‌های تودرتو، میزان کاهش میانگین 36/2% برای داده‌های نقطه‌وارسی داشته است. همچنین زمان کلی اجرا را 27/2% بهبود بخشیده است. این روش در سیستم‌هایی که با مشکلات قطعی برق مواجه هستند می‌تواند کاربردی باشد. مانند سیستم‌هایی که منبع انرژی آنها از طبیعت است و یا بدون باتری کار می‌کنند و احتمال قطعی برق در آنها زیاد است.

### 5.3.2. روش بازیابی محلی برای برنامه‌های رابط فرستادن پیام

برای استفاده از خوشه‌های متشکل از چند کامپیوتر مشخصه رابط فرستادن پیام<sup>2</sup> در برنامه‌ها کاربرد دارد. در تکنیک‌های مبتنی بر بازگشت محلی با افزایش اجزا و ابزارهای محاسباتی، تاثیر منفی در واحد متوسط زمان میان‌خرابی‌ها مشاهده شده است. کار تحقیقی گروه Kiril Dichev برای کاهش سربار در این‌گونه روش‌ها و همچنین بهبود مصرف انرژی در این‌گونه سیستم‌ها است [47]. نویسندگان با ارائه الگوریتمی تنها به کاهش هزینه‌های نقطه‌وارسی در فرآیندهای فعال و خراب پرداخته‌اند. دلیل این کار این است که تنها این فرآیندها در پروسه بازیابی دخیل هستند. در عملیات بازیابی برای فرآیندی که دچار اشکال شده است فرآیندهای مجموعه فعال با ارسال پیام به فرآیند کمک می‌کنند تا بازیابی شود. با محدود کردن تعداد فرآیندهای عملیات بازیابی شاهد کاهش سرباری خواهیم بود که برای ما بهبود مصرف انرژی را هم به همراه دارد. روش این گروه بر پایه ثبت پیام<sup>3</sup> است که محدودیت کاربردی روش‌های پیشین را

فرآیند ایجاد نقطه‌وارسی را با محاسبات برنامه همگام می‌کند تا کارایی سیستم را افزایش دهد. همچنین تنها تغییرات نسبت به آخرین نقطه‌وارسی را ذخیره می‌کند.

### 3.3.2. CRAFT. یک کتابخانه جهت پیاده‌سازی نقطه‌وارسی

#### در برنامه‌ها

پیاده‌سازی نقطه‌وارسی در سطح برنامه‌ها دارای پیچیدگی در شناسایی خرابی‌ها است. برای پیاده‌سازی یک فرآیند نقطه‌وارسی آسان که تحمل‌پذیری خطاها را برای ما فراهم کند، گروه تحقیقاتی Shahzad کتابخانه CRAFT را ارائه کرده‌اند [45]. این کتابخانه به زبان ++C توسعه داده شده است. در CRAFT به صورت ساده یک شی از نقطه‌وارسی ساخته می‌شود و تمام داده‌های مرتبط به آن توسط تابع افزودن اضافه می‌شود. زمانی که تمام داده‌های مرتبط اضافه شد، شی نقطه‌وارسی ذخیره می‌شود. در CRAFT کاربران می‌توانند ساختار داده‌های خود را تعریف کنند. این کتابخانه از نقطه‌وارسی آسنکرون و کتابخانه SCR پشتیبانی می‌کند تا سربار نقطه‌وارسی و بازیابی را کاهش دهد [27]. این امر با نقطه‌وارسی سطح گره امکان‌پذیر است. سطح گره سه حالت گره محلی، مشارکتی و مشارکتی XOR را شامل می‌شود. در صورت خرابی یک گره، سطح مشارکتی این اجازه را می‌دهد تا داده‌ها را از همسایه گره خراب شده بازیابی کند.

### 4.3.2. کاشی‌کاری حلقه‌ها با آگاهی از نقطه‌وارسی

در مقاله [46]، به سربار نقطه‌وارسی در اجرای حلقه‌های تودرتو پرداخته شده است. در این حلقه‌ها هر واحد محاسباتی در یک گام حلقه محاسبه می‌شود. هر گام حلقه به دیگر گام‌ها وابستگی داده‌ای دارد. برای ایجاد نقطه‌وارسی در این حلقه‌ها بایستی محاسبات به دست آمده در هر گام حلقه که درون حافظه فرار نگهداری می‌شود، به حافظه‌ای غیرفرار منتقل شود. با وجود تودرتو بودن حلقه‌ها و وابستگی میان آنها در روش عادی نیاز به گرفتن نقطه‌وارسی از تمامی گام‌ها است که سربار زیادی دارد.

<sup>1</sup> Tiling

<sup>2</sup> Message Passing Interface (MPI)

<sup>3</sup> Message Logging

وابسته به آخرین شکست<sup>1</sup> و نقطه‌واری وابسته به میانگین شکست<sup>2</sup> شد [50].

در الگوریتم اول، هدف کاهش سربار ناشی از نقطه‌های واری بیش از اندازه در محیط‌های به نسبت پایدار توزیع شده است. این الگوریتم نقطه‌های واری غیرضروری مربوط به فرآیند را با توجه به زمان تقریبی اجرای آن و تعداد خرابی محیط اجرای آن، حذف می‌کند. در الگوریتم نقطه‌واری وابسته به میانگین شکست برخلاف الگوریتم قبلی، به صورت پویا تعداد فرآیند نقطه‌واری را که در ابتدا تعیین شده است، تغییر می‌دهد. این تغییر برای مقابله با فاصله‌های نامناسب نقطه‌های واری انجام می‌شود. هر زمان که یک نقطه‌واری ایجاد شود، فاصله نقطه‌واری خود را متناسب با زمان اجرای باقی‌مانده و میانگین فاصله خرابی تغییر می‌دهد. بدیهی است که با افزایش فاصله بین نقطه‌های واری تعداد آنها را کاهش می‌دهد و از این رو سربار فرآیند نقطه‌واری کاهش پیدا می‌کند. هر دو الگوریتم پیشنهاد شده سربار زمان اجرا را کاهش می‌دهند.

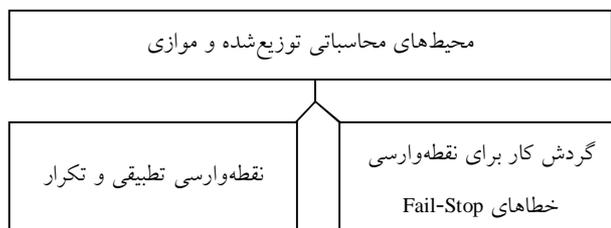
#### 2.4.2. گردش کار برای نقطه‌واری خطاهای توقف/شکست

در سیستم‌های محاسبات موازی، روش‌های پیشین All-Checkpoint با ذخیره تمام داده‌های خروجی باعث ایجاد سربار زیادی در سیستم می‌شدند [51]، [52]. محققان با ایجاد شرط‌هایی بر روی این داده‌های خروجی سعی در کاهش سربار نقطه‌واری داشته‌اند. مقاله [53] با استفاده از ارتباط و وابستگی بین فرآیندها سعی در بهبود سربار دارد. این روش Checkpoint-Some نام دارد. در این روش با استفاده از سری گراف‌های موازی کمینه، امکان محاسبه مجموعه‌ای از کارهای مستقل در یک پردازنده و ارتباط با پردازنده‌های دیگر را فراهم می‌سازد. با ایجاد نقطه‌واری از کارهای خروجی یک Superchain وابستگی‌های متقاطع از بین می‌رود. این روش یک زمان‌بندی را برای ایجاد نقطه‌واری معرفی می‌کند تا میان داده‌های نقطه‌واری وابستگی وجود نداشته باشد. با ایجاد

ندارد. این الگوریتم باعث می‌شود تا در مرحله بازیابی تا حدود 50% انرژی در بین پردازنده‌ها نسبت به روش‌های بر پایه ثبت پیام صرفه‌جویی شود.

#### 4.2. روش‌های نقطه‌واری در محیط‌های محاسباتی توزیع شده

در محیط‌های محاسباتی توزیع شده استفاده از نقطه‌واری یک امر مهم است، زیرا این رویکرد باعث می‌شود سیستم قابلیت تحمل‌پذیری اشکال پیدا کند. به دلیل اجرای طولانی مدت نرم‌افزارها در این گونه محیط‌ها، ایجاد یک تصویر کلی از حالت نرم‌افزار در لحظه ایجاد نقطه‌واری امری ضروری است. این نقطه‌واری باعث می‌شود تا در صورت بروز اشکال در سیستم، نرم‌افزار بتواند از آخرین نقطه‌واری خود عملیات را ادامه دهد. بدین ترتیب از کارایی سیستم و قابلیت اطمینان آن در برابر اشکال‌ها مراقبت شده است [48]. روش‌های این دسته‌بندی دارای ویژگی مشترک طولانی بودن مدت زمان اجرای برنامه‌های خود هستند. روش‌های کاهش سربار نقطه‌واری در محیط‌های محاسباتی توزیع شده و موازی در شکل (5) نشان داده شده است.



شکل (5): روش‌های کاهش سربار نقطه‌واری در محیط‌های محاسباتی توزیع شده و موازی

#### 1.4.2. نقطه‌واری تطبیقی و تکرار، روشی کارآمد برای تحمل اشکال

در سیستم‌های توزیع شده، طبق مطالعات و آزمایش‌ها محققان به این نتیجه رسیده‌اند که رویکردهای انطباقی کارایی سیستم‌ها را بسیار بهبود می‌بخشند [49]. کاهش سربار نقطه‌واری در این سیستم‌ها هدفی بود که باعث پیشنهاد الگوریتم‌های نقطه‌واری

<sup>1</sup> Last Failure Dependent Checkpointing

<sup>2</sup> Mean Failure Dependent Checkpointing

معیوب سیستم از دوره‌های زمانی یکنواخت برای ایجاد نقطه‌وارسی استفاده می‌کنند. طول این فواصل در قسمت آفلاین و پیش از اجرای سیستم تعیین می‌شود. با رخداد اولین اشکال، سیستم از نقطه‌وارسی غیریکنواخت به حالت نقطه‌وارسی یکنواخت تغییر حالت می‌دهد. این روش به طور میانگین 62% تعداد نقطه‌های واری را کاهش می‌دهد. سربرار کاهش یافته با این روش نسبت به روش‌های گذشته 14% اعلام شده است. سایر روش‌هایی که مورد مقایسه با روش بالا قرار گرفته‌اند، شامل نقطه‌وارسی منظم بهینه<sup>1</sup> [58]، نقطه‌وارسی نامنظم<sup>2</sup> [59] و نقطه‌وارسی منظم انطباقی<sup>3</sup> [60] هستند.

### 2.5.2. FREM، روش کلی راه‌اندازی مجدد سریع برای

#### نقطه‌وارسی / راه‌اندازی مجدد

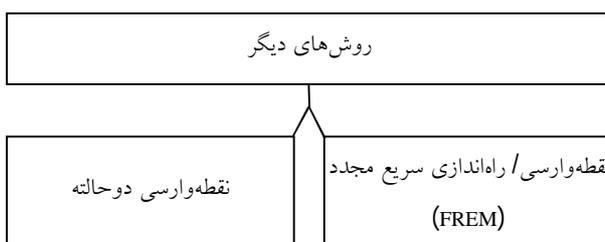
روش‌هایی که تاکنون مرور کردیم، کاهش سربرار در عملیات نقطه‌وارسی را بررسی می‌کردند. در مقاله [61]، نویسندگان بر بهبود عملیات راه‌اندازی مجدد در تمام فرآیندهای نقطه‌وارسی تمرکز کرده‌اند. آنها تاثیر سربرار ناشی از راه‌اندازی مجدد را بررسی کرده‌اند و روش پیشنهادی خود را ارائه کرده‌اند.

روش FREM با همپوشانی بازیابی برنامه با تصویر نقطه‌وارسی آن، تاخیر راه‌اندازی مجدد را کاهش می‌دهد. این روش به بهبود کارایی هر سیستم مبتنی بر نقطه‌وارسی و راه‌اندازی مجدد کمک می‌کند. این روش از دو بخش Postcheckpoint Tracking (CT) و Fast Restarting (FR) تشکیل شده است. بخش CT خود از دو زیربخش عملیات ایجاد نقطه‌وارسی سراسری و رصد مجموعه مورد تغییر تشکیل شده است. این بخش‌ها به صورت هم‌پوشانی اجرا می‌شوند تا در صورتی که در پنجره ردیابی اشکالی رخ دهد، فرآیند از سر گرفته شود. در بخش FR هدف FREM دستیابی به همپوشانی بین اجرای فرآیند و بازیابی نقطه‌وارسی است. در نتیجه تاخیر راه‌اندازی مجدد به زمان بازیابی با اطلاعات نقطه‌وارسی را کاهش می‌دهد. روش پیشنهادی این مقاله ویژگی‌های کارایی، عمومیت و پنهان بودن

این‌گونه نقطه‌های واری در صورت شکست نیازی به اجرای دوباره نیست و از آخرین نسخه طبق وابستگی‌های قبلی می‌توان استفاده کرد. روش پیشنهادی Checkpoint-Some از روش Checkpoint-All عملکرد بهتری داشته و باعث کاهش قابل توجه در سربرار نقطه‌وارسی شده است.

### 5.2. روش‌های دیگر

در این بخش چند روش کاهش سربرار نقطه‌وارسی جدا از دسته‌بندی‌های دیگر معرفی می‌شوند. استفاده از راهبردهای پیشبینی و زمانبندی پویا می‌تواند باعث بهبود اجرای فرآیند نقطه‌وارسی شود. برخی از این راهبردها را می‌توان در چندین مطالعه [54] - [56] مشاهده کرد. روش‌های بررسی شده در شکل (6) قابل مشاهده است.



شکل (6): روش‌های دیگر کاهش سربرار نقطه‌وارسی

### 1.5.2. نقطه‌وارسی دو حالتی برای تحمل پذیری اشکال و

#### بهره‌وری انرژی

استفاده از نقطه‌های واری منظم هنگام اطمینان از سلامت سیستم تنها باعث اتلاف انرژی و زمان می‌شود. این ایده بنای ارائه روش Two-State Checkpointing است [57]. طبق بررسی نویسندگان کاهش تعداد نقطه‌های واری در دوره بدون خطا سیستم می‌تواند با کاهش میانگین تعداد نقطه‌های واری در دوره‌های زمانی، باعث کاهش میانگین زمان اجرا و مصرف انرژی شود. آنها روش خود را در دو مرحله آفلاین و آنلاین معرفی کرده‌اند. در مرحله آفلاین که در زمان طراحی انجام می‌شود، سیستم را برای زمان اجرا به دو قسمت زمانی بدون عیب و معیوب تقسیم می‌کنند. از دوره‌های زمانی غیریکنواخت برای زمان بدون عیب استفاده می‌شود. در مقابل برای حالت‌های

<sup>1</sup> Optimal Uniform Checkpointing

<sup>2</sup> NonUniform Checkpointing

<sup>3</sup> Adaptive Uniform Checkpointing

شده، پیشنهاد می‌شود به‌جای پیاده‌سازی الگوریتم‌های بازگشتی که دقت بالایی ندارند، از الگوریتم‌های پیش‌بینی هوش مصنوعی کمک گرفته شود. در روش‌های مبتنی بر نقطه‌وارسی افزایشی، روش‌های سعی در به حداقل رساندن اطلاعات هر نقطه‌وارسی داشته‌اند تا بتوانند به کاهش سربار نقطه‌وارسی کمک کنند. برای مثال روش نقطه‌وارسی افزایشی با توجه به ساختار داده [35]، با ویژگی‌های مجموعه‌ها و بررسی اعتبار دستورات سعی در حذف دستورات اضافی داشته است. در روش Survive هم با ایجاد لیست اشاره‌گرها سعی در کاهش اطلاعات ذخیره شده در هر نقطه‌وارسی داشته است [40]. بررسی‌ها نشان می‌دهد می‌توان از ویژگی‌های دیگر سیستمی مانند کدهای برنامه برای پیش‌بینی اطلاعات بدون استفاده و یا تکراری استفاده کرد. با حذف این داده‌های اضافی می‌توان سربار نقطه‌وارسی را در روش‌های مبتنی بر نقطه‌وارسی افزایشی بیش‌ازپیش کاهش داد. در روش‌های سطح برنامه، کاهش سربار نقطه‌وارسی با استفاده از ویژگی‌های برنامه‌ها و دستورات آنها قابل تحقق است. در روش کاشی‌کاری حلقه‌ها، ویژگی وابستگی میان داده‌ها در گام‌های حلقه‌های تودرتو باعث آن شد که با تغییر دستورات برنامه شاهد کاهش تعداد نقطه‌های واری موردنیاز باشیم [46]. طبق بررسی‌ها پیشنهاد می‌شود بر روی دیگر اجزا دستورات برنامه‌ها مانند پیش‌بینی دستورات شرطی تمرکز شود تا از ایجاد نسخه حجیم در نقطه‌وارسی جلوگیری شود. در جدول (1) مقایسه‌ای کلی میان همه روش‌ها با توجه به معیار بهبود عملکرد بیان شده است. مزایا و معایب گفته شده در این جدول به صورت تحلیل نظری از هر روش بیان شده است.

#### 4. نتیجه‌گیری

در این کار تلاش شده تا مروری بر روش‌های کاهش سربار نقطه‌وارسی ارائه شود. تمرکز بر روی روش‌های کاهش سربار در نقطه‌های واری هماهنگ، نقطه‌وارسی در سطح سیستم، نقطه‌وارسی در سطح برنامه و نقطه‌وارسی در سیستم‌های توزیع شده بوده است. با توجه به کاربرد گسترده نقطه‌وارسی، این زمینه موضوعی باز برای تحقیق و پژوهش‌های آینده است.

از دید کاربر را دارد. از طرفی به دلیل انجام عملیات رصدکردن و بازیابی سریع، مقداری سربار ایجاد می‌کند که از نظر نویسندگان، به دلیل مزایای این روش می‌توان از آن چشم‌پوشی شود.

### 3. نتایج و پیشنهادهای آتی

کاهش سربار نقطه‌وارسی هدف مشترک همه روش‌های بررسی شده بوده است. در نقطه‌های واری هماهنگ شده برای کاهش سربار بایستی اصل سازگاری سیستم را هم رعایت کرد. در روش نقطه‌وارسی چندسطحی بدون دیسک [13]، نقطه‌وارسی با وجود بازیابی حالت سازگار در زمان برخورد با چندین خطا، دارای سربار برای سیستم است. این سربار ممکن است از نظر انرژی و پیچیدگی پیاده‌سازی نقطه‌وارسی چندسطحی پیش‌آید؛ اما در مقابل این هزینه، افزایش عملکرد سیستم را با مقاومت در برابر چندین خرابی به ارمغان می‌آورد. روش نقطه‌وارسی آنلاین دوسطحی بهبودیافته [17] هم در راستای روش قبلی به حل چالش‌های داخل هر سطح نقطه‌وارسی پرداخته است. هزینه‌های ناشی از پیاده‌سازی نقطه‌وارسی چندسطحی بایستی نسبت به عملکرد دریافتی سنجیده شوند تا مقرون‌به‌صرفه بودن آنها حاصل شود. استفاده از ویژگی‌های سیستم‌ها در راستای کاهش سربار نقطه‌وارسی دلیلی بر معرفی روش‌های فعال نقطه‌وارسی شده است. روش نقطه‌وارسی فعال استفاده شده در سیستم‌های پردازش ابری [20]، از ویژگی دمای پردازنده برای تعیین زمان نقطه‌وارسی استفاده کرده است. ایجاد نقطه‌وارسی در مواقع حساس سیستم و مستعد خرابی بالا باعث شده تا از ایجاد نقطه‌وارسی بهبود یافته در زمان عاری از خطا جلوگیری شود. پیشنهاد می‌شود برای تکمیل این پژوهش‌ها، می‌توان از ویژگی‌های دیگر سیستم‌ها مانند ولتاژ ورودی به پردازنده و یا شدت عملکرد سیستم خنک‌کننده برای ایجاد نقطه‌وارسی استفاده شود. زمان‌بندی بهینه ایجاد نقطه‌وارسی با ویژگی‌های دیگری از سیستم‌ها مانند میزان ترافیک ورودی و خروجی سیستم، تعریف آستانه زمانی برای ایجاد نقطه‌وارسی و یا پیش‌بینی زمان ایجاد نقطه‌وارسی قابل پیاده‌سازی است. با توجه به اطلاعات و داده‌های موجود در سیستم‌ها برای ویژگی‌های ذکر

جدول (1). مقایسه روش های کاهش سربار نقطه‌وارسی

منبع	سال انتشار	معایب	مزایا	نام روش
[32]	2022	طراحی مجدد کنترل‌کننده حافظه نیاز به پیاده‌سازی بخش‌های جدید سخت‌افزاری	بهبود عملکرد با کاهش تداخل نقطه‌وارسی و برنامه‌های در حال اجرا	بهبود موازی‌سازی سطح بانک برای نقطه‌وارسی در حافظه
[47]	2022	نیاز به ابزار دقیق برای کدنویسی فقط تماس‌های نقطه‌به‌نقطه در نمونه اولیه ثبت شده‌اند	بهبود مصرف انرژی قابل‌اعمال بر روی اکثر سیستم‌های محاسباتی با کارایی بالا	روش بازیابی محلی برای برنامه‌های رابط فرستادن پیام
[21]	2014	پیچیدگی سیستم نقطه‌وارسی به دلیل پیاده‌سازی و مدیریت سیاست‌های تطبیق	صرفه‌جویی در مصرف انرژی با تأثیر کمتر در کارایی قابلیت تطبیق	ایجاد یا عدم ایجاد نقطه‌وارسی
[24]	1992	افزایش استفاده از حافظه پردازنده سربار مربوط به محاسبات چک‌سام	کاهش سربار نقطه‌وارسی تا 60 درصد نسبت به روش‌های سنتی بهبود عملکرد کاهش پهنای باند انتها به انتها	HeteroCheckpoint نقطه‌وارسی کارآمد برای شتاب‌دهنده‌ها
[43]	2012	پیچیدگی بیشتر نسبت به نقطه‌های واری ساده	کاهش قابل‌توجه اندازه نقطه‌وارسی حداقل سربار عملکرد در مقایسه با نقطه‌وارسی فشرده نشده (کمتر از 2 درصد) مقیاس‌پذیری در سیستم‌های بزرگ	کاهش اندازه فایل‌های نقطه‌وارسی در سطح برنامه
[44]	2012	افزایش پیچیدگی به دلیل اضافه‌شدن مدیریت حافظه و تکنیک‌های بهینه‌سازی	کاهش زمان در نقطه‌وارسی و بازیابی (تا 15 برابر بهبود) وجود نقطه‌وارسی دیفرانسیلی و کاهش هزینه تا 2,6 درصد پشتیبانی از معماری‌های متنوع و پیچیده بهبود عملکرد کلی	راه‌اندازی مجدد نقطه‌وارسی برای برنامه‌های HPC ناهمگن
[20]	2018	پارامترهای دریافتی در این روش بایستی قابل‌اتکا باشند	بهبود زمان کلی اجرا کاهش خرابی منجر به بازیابی	استفاده از روش فعال نقطه‌وارسی
[57]	2016	زمان مواجهه سیستم با خطا و تغییر نوع نقطه‌وارسی همچنان شاهد کاهش عملکرد هستیم	افزایش کارایی کلی سیستم با کاهش تعداد نقطه‌های واری کاهش مصرف انرژی	نقطه‌وارسی دوحالتی برای تحمل‌پذیری اشکال و بهره‌وری انرژی
[39]	2019	در ایجاد نقطه‌وارسی از صفحات بیکار حافظه کند عمل می‌کند	کاهش اطلاعات ذخیره شده در هر نقطه‌وارسی بهبود مصرف انرژی	نقطه‌وارسی سبک در حافظه‌های مجازی موبایل
[17]	2017	در روش آنلاین پیشنهاد شده، بهبودها مانند روش آفلاین است	بهبود زمان کلی اجرا تحمل‌پذیری اشکال در سطوح مختلف	روش نقطه‌وارسی آنلاین دوسطحی بهبودیافته
[40]	2017	ذخیره اشاره‌گرها سربار حافظه دارد	افزایش قابلیت اطمینان سیستم افزایش اولویت دیگر برنامه‌ها به حافظه غیرفرار	Survive، روش نقطه‌وارسی افزایشی بر مبنای اشاره‌گر

ادامه جدول (1). مقایسه روش‌های کاهش سربار نقطه‌وارسی

نام روش	مزایا	معایب	سال انتشار	منبع
کاشی‌کاری حلقه‌ها با آگاهی از نقطه‌وارسی	بهبود زمان کلی اجرا کاهش داده‌های ذخیره شده در نقطه‌وارسی	نیاز به تغییر کد برنامه‌ها دارد	2019	[46]
گردش کار برای نقطه‌وارسی خطاهای توقف/شکست	زمانی که نرخ شکست‌ها بالا باشد عملکرد بهتری دارد	در شرایط پایین‌بودن نرخ شکست، روش‌های قبلی عملکرد بهتری داشته‌اند	2018	[53]
CRAFT، کتابخانه‌ای جهت پیاده‌سازی نقطه‌وارسی در برنامه‌ها	ساده‌سازی پیاده‌سازی نقطه‌وارسی در سطح برنامه داشتن روش‌های مختلف گرفتن نقطه‌وارسی و حق انتخاب از بین آنها کاهش سربار نسبت به کتابخانه‌های نقطه‌وارسی در سطح سیستم	باتوجه به اینکه یک کتابخانه عمومی است بر اساس نیازهای سیستم باید تغییراتی را به آن اعمال کرد	2019	[45]
روش مبتنی بر گراف در سیستم‌های موبایل	کاهش زمان گرفتن نقطه‌وارسی کاهش نقاط واری	افزایش حجم پیام‌های سیستم افزایش سربار محاسباتی هر گره	2013	[19]
SCR، کتابخانه‌ای مقیاس‌پذیر برای نقطه‌وارسی	کاهش فرکانس گرفتن نقطه‌وارسی افزایش سرعت شروع مجدد سیستم	نیاز به فضا در کش برای ذخیره‌سازی نقاط واری در هر گره	2010	[27]
نقطه‌وارسی تطبیقی و تکرار، روشی کارآمد برای تحمل اشکال	گرفتن نقطه‌وارسی باتوجه به پارامترهای مختلف که باعث کاهش گرفتن نقاط واری می‌شود	افزایش سربار محاسباتی برای محاسبه الگوریتم‌ها	2009	[50]
انتخاب بهینه نقطه‌وارسی با افزونگی دو مدولار	افزایش کارایی پردازنده تضمین راه‌حل بهینه در ایجاد نقطه‌وارسی در مرز فعالی‌ها	کند بودن نسبت به الگوریتم‌های پایه‌ای این روش	2015	[22]
تصویر لحظه‌ای درخواستی، یک سیستم نسخه‌سازی کارآمد	بهبود کارایی سیستم و توان عملیاتی کاهش نوشتن‌های غیرضروری در نقطه‌وارسی بهبود مصرف انرژی	مشکل پیچیدگی حافظه به دلیل نیاز آن به حافظه برای ردیابی تصاویر لحظه‌ای	2013	[33]
نقطه‌وارسی چندسطحی بدون دیسک	بهبود زمان اجرا در پیاده‌سازی سطوح دو و سه عملکرد بهتر نسبت به نقطه‌وارسی تک‌سطحی هنگام افزایش نرخ شکست	افزایش سطوح نقطه‌وارسی باعث افزایش هزینه‌های پیاده‌سازی می‌شود	2013	[13]
نقطه‌وارسی افزایشی، یک رویکرد بر مبنای ساختار داده	در صورتی که تعداد دستورات ما بین دو نقطه‌وارسی کمتر از 100 دستور باشد، بهبود عملکرد و کاهش سربار دارد	وابسته به تعداد دستورات افزودن/حذف کردن در میان نقطه‌های واری است	2017	[35]

باعث کاهش عملکرد کلی سیستم‌ها نشود. هدف اصلی روش‌هایی مانند نقطه‌وارسی چندسطحی بدون دیسک، انتخاب بهینه نقطه‌وارسی با افزونگی، بهبود موازی‌سازی سطح بانک حافظه، تصویر لحظه‌ای درخواستی، نقطه‌وارسی افزایشی بر

افزایش قابلیت اطمینان سیستم‌ها و عدم اتلاف کارایی به دلیل خرابی‌ها، دلایل اصلی استفاده از یک روش نقطه‌وارسی است. همه تکنیک‌های بررسی شده در این کار در تلاش بوده‌اند تا یک روش بهینه با سربار پایین معرفی کنند تا عملیات نقطه‌وارسی

مقایسه کند، امکان‌پذیر نیست. همچنین روش‌های گفته شده روی سیستم‌های مختلف و با استفاده از ابزارهای شبیه‌سازی متفاوت پیاده‌سازی شده است. با این وجود ملاک بررسی ما نتایج آزمایش‌ها و پیاده‌سازی‌های مختلف نویسندگان بوده است. ادعای نویسندگان درباره میزان بهبود روش پیشنهادی، در پایان بررسی هر روش گفته شده است.

تعارض منافع: نویسندگان اعلام می‌کنند که هیچ تعارض منافی ندارند.

مبنای ساختار داده و کاشی‌کاری حلقه‌ها با آگاهی از نقطه‌وارسی بهبود عملکرد سیستم با کاهش سربار نقطه‌وارسی بوده است. با بررسی کارهای مطالعاتی انجام شده مشاهده می‌شود که هدف‌های اصلی مطالعات، کاهش تعداد نقطه‌های واریسی، بهینه‌کردن فرآیند ذخیره‌سازی داده‌ها و کاهش حجم داده‌های هر نقطه‌وارسی بوده است. این‌ها همه به‌منظور کاهش سربار کلی نقطه‌وارسی و افزایش کارایی سیستم‌ها انجام شده است. روش‌های بررسی شده که در شکل (1) قابل مشاهده هستند، سعی داشته‌اند با استفاده از معیارهای مختلفی سربار نقطه‌وارسی را کاهش دهند. به دلیل تنوع در کاربرد نقطه‌وارسی، تعیین یک معیار که بتواند همه این روش‌ها را به صورت دقیق با یکدیگر

## مراجع

- [1] I.P. Egwuotuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *J. Supercomput.*, vol. 65, pp. 1302-1326, 2013, doi: 10.1007/s11227-013-0884-0.
- [2] N.H. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme," *IEEE Trans. Comput.*, vol. 46, no. 8, pp. 942-947, 1997, doi: 10.1109/12.609281.
- [3] D. Loreti, M. Artioli, and A. Ciampolini, "Rollback-free recovery for a high performance dense linear solver with reduced memory footprint," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 7, pp. 1307-1319, 2024, doi: 10.1109/TPDS.2024.3400365.
- [4] H. Benkaouha, A. Abdelli, J. Ben-Othman, Y. Zaffoune, and L. Mokdad, "Distributed implementation of a stable storage for MANET checkpointing protocols," in *Proc. Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Paphos, Cyprus, 2016, pp. 672-677, doi: 10.1109/IWCMC.2016.7577137.
- [5] K. Hwang, H. Jin, E. Chow, C.-L. Wang, and Z. Xu, "Designing SSI clusters with hierarchical checkpointing and single I/O space," *IEEE Concurrency*, vol. 7, no. 1, pp. 60-69, 1999, doi: 10.1109/4434.749136.
- [6] X. Cui et al., "A snake addressing scheme for phase change memory testing," *Sci. China Inf. Sci.*, vol. 59, no. 10, p. 102401, 2016, doi: 10.1007/s11432-015-5437-0.
- [7] G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill, "Automated application-level checkpointing of MPI programs," *ACM SIGPLAN Notices*, vol. 38, no. 10, pp. 84-94, 2003, doi: 10.1145/966049.781513.
- [8] I.P. Egwuotuoha et al., "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *J. Supercomput.*, vol. 65, pp. 1302-1326, 2013, doi: 10.1007/s11227-013-0884-0.
- [9] E.N. Elnozahy, L. Alvisi, Y.-M. Wang, and D.B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375-408, 2002, doi: 10.1145/568522.568525.
- [10] G. Cao and M. Singhal, "On coordinated checkpointing in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 12, pp. 1213-1225, 1998, doi: 10.1109/71.737697.
- [11] M.T. Raza, Z. Tan, A. Tufail, and F.M. Anwar,

- “LTE NFV rollback recovery,” *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 2468-2477, 2022, doi: 10.1109/TNSM.2022.3182008.
- [12] J. Plank, K. Li, and M. Puening, “Diskless checkpointing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 10, pp. 972-986, 1998, doi: 10.1109/71.730527.
- [13] D. Hakkarinen and Z. Chen, “Multilevel diskless checkpointing,” *IEEE Trans. Comput.*, vol. 62, no. 4, pp. 772-783, 2013, doi: 10.1109/TC.2012.17.
- [14] L. Gomez, A. Nukada, N. Maruyama, F. Cappello, and S. Matsuoka, “Low-overhead diskless checkpoint for hybrid computing systems,” in *Proc. Int. Conf. High Perform. Comput.*, 2010, pp. 1-10, doi: 10.1109/HIPC.2010.5713163.
- [15] L.A.B. Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, “Distributed diskless checkpoint for large scale systems,” in *Proc. 10th IEEE/ACM Int. Conf. Cluster Cloud Grid Comput.*, 2010, pp. 63-72, doi: 10.1109/CCGRID.2010.40.
- [16] H. Li, L. Pang, and Z. Wang, “Two-level incremental checkpoint recovery scheme for reducing system total overheads,” *PLOS ONE*, vol. 9, no. 8, p. e104591, 2014, doi: 10.1371/journal.pone.0104591.
- [17] S. Di, Y. Robert, F. Vivien, and F. Cappello, “Toward an optimal online checkpoint solution under a two-level HPC checkpoint model,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 244-259, 2017, doi: 10.1109/TPDS.2016.2546248.
- [18] S.S. Manvi and P. Venkataram, “Applications of agent technology in communications: A review,” *Comput. Commun.*, vol. 27, pp. 1493-1508, 2004, doi: 10.1016/j.comcom.2004.05.011.
- [19] R. Singh and M. Dave, “Antecedence graph approach to checkpointing for fault tolerance in mobile agent systems,” *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 247-258, 2013, doi: 10.1109/TC.2011.235.
- [20] J. Liu et al., “Using proactive fault-tolerance approach to enhance cloud service reliability,” *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1191-1202, 2018, doi: 10.1109/TCC.2016.2567392.
- [21] N. El-Sayed and B. Schroeder, “To checkpoint or not to checkpoint: Understanding energy-performance-I/O tradeoffs in HPC checkpointing,” in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Madrid, Spain, 2014, pp. 93-102, doi: 10.1109/CLUSTER.2014.6968778.
- [22] S.-H. Kang, H.-W. Park, S. Kim, H. Oh, and S. Ha, “Optimal checkpoint selection with dual-modular redundancy hardening,” *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 2036-2048, 2015, doi: 10.1109/TC.2014.2349492.
- [23] K.B. Ferreira, R. Riesen, R. Brightwell, P. Bridges, and D. Arnold, “Recent advances in the Message Passing Interface,” *Int. J. High Perform. Comput. Appl.*, 2011, doi: 10.1177/1094342014549273.
- [24] S. Kannan, N. Farooqui, A. Gavrilovska, and K. Schwan, “HeteroCheckpoint: Efficient checkpointing for accelerator-based systems,” in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Atlanta, GA, USA, 2014, pp. 738-743, doi: 10.1109/DSN.2014.76.
- [25] K. Iskra, J.W. Romein, K. Yoshii, and P. Beckman, “ZOID: I/O-forwarding infrastructure for petascale architectures,” in *Proc. 13th ACM SIGPLAN Symp. Principl. Pract. Parallel Prog.*, 2008, pp. 153-162, doi: 10.1145/1345206.1345230.
- [26] E. Gelenbe, “A model of roll-back recovery with multiple check-points,” in *Proc. 2nd Int. Conf. Softw. Eng. (ICSE'76)*, 1976, pp. 251-255, doi: 10.5555/800253.807684.
- [27] A. Moody, G. Bronevetsky, K. Mohror, and B.R. de Supinski, “Design, modeling, and evaluation of a scalable multi-level checkpointing system,” in *Proc. ACM/IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, New Orleans, LA, USA, 2010, pp. 1-11, doi: 10.1109/SC.2010.18.
- [28] X. Tang et al., “An efficient in-memory checkpoint method and its practice on fault-tolerant HPL,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 758-771, 2018, doi: 10.1109/TPDS.2017.2781257.
- [29] H. Cho, E. Cheng, T. Shepherd, C.-Y. Cher, and S.

- Mitra, "System-level effects of soft errors in uncore components," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1497-1510, 2017, doi: 10.1109/TCAD.2017.2651824.
- [30] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic, "Optimizing checkpoints using NVM as virtual memory," in *Proc. 27th Int. Symp. Parallel Distrib. Process. (IPDPS)*, 2013, pp. 29-40, doi: 10.1109/IPDPS.2013.69.
- [31] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2012, pp. 367-376.
- [32] X. Liao, Z. Zhang, H. Liu, and H. Jin, "Improving bank-level parallelism for in-memory checkpointing in hybrid memory systems," *IEEE Trans. Big Data*, vol. 8, no. 2, pp. 289-301, 2022, doi: 10.1109/TBDATA.2018.2865964.
- [33] E. Lee, J.E. Jang, T. Kim, and H. Bahn, "On-demand snapshot: An efficient versioning file system for phase-change memory," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2841-2853, 2013, doi: 10.1109/TKDE.2013.35.
- [34] Btrfs, "Main Page," 2013. [Online]. Available: <http://btrfs.wiki.kernel.org>
- [35] C.-Y. Lin, L.-C. Wang, and S.-P. Chang, "Incremental checkpointing for fault-tolerant stream processing systems: A data structure approach," *IEEE Trans. Emerging Topics Comput.*, vol. 10, no. 1, pp. 124-136, 2022, doi: 10.1109/TETC.2020.2986487.
- [36] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "Aries: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," *ACM Trans. Database Syst.*, vol. 17, no. 1, pp. 94-162, 1992, doi: 10.1145/128765.128770.
- [37] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker, "Rethinking main memory OLTP recovery," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Chicago, IL, USA, 2014, pp. 604-615.
- [38] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Trans. Storage*, vol. 8, no. 4, pp. 14:1-14:25, 2012, doi: 10.1145/2385603.2385607.
- [39] K. Zhong et al., "Towards fast and lightweight checkpointing for mobile virtualization using NVRAM," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1421-1433, 2019, doi: 10.1109/TPDS.2018.2886906.
- [40] A. Mirhosseini, A. Agrawal, and J. Torrellas, "Survive: Pointer-based in-DRAM incremental checkpointing for low-cost data persistence and rollback-recovery," *IEEE Comput. Archit. Lett.*, vol. 16, no. 2, pp. 153-157, 2017, doi: 10.1109/LCA.2016.2646340.
- [41] M. Prvulovic, Z. Zhang, and J. Torrellas, "ReVive: Cost-effective architectural support for rollback recovery in shared-memory multiprocessors," in *Proc. 29th Annu. Int. Symp. Comput. Archit. (ISCA)*, Anchorage, AK, USA, 2002, pp. 111-122, doi: 10.1109/ISCA.2002.1003567.
- [42] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 3, pp. 212-226, 2009, doi: 10.1177/1094342009106189.
- [43] I. Cores, G. Rodriguez, M.J. Martin, and P. Gonzalez, "Reducing application-level checkpoint file sizes: Towards scalable fault tolerance solutions," in *Proc. IEEE 10th Int. Symp. Parallel Distrib. Process. Appl. (ISPA)*, Leganes, Spain, 2012, pp. 371-378, doi: 10.1109/ISPA.2012.55.
- [44] K. Parasyris, K. Keller, L. Bautista-Gomez, and O. Unsal, "Checkpoint restart support for heterogeneous HPC applications," in *Proc. 20th IEEE/ACM Int. Symp. Cluster Cloud Internet Comput. (CCGRID)*, Melbourne, VIC, Australia, 2020, pp. 242-251, doi: 10.1109/CCGrid49817.2020.00-69.
- [45] F. Shahzad, J. Thies, M. Kreutzer, T. Zeiser, G. Hager, and G. Wellein, "CRAFT: A library for easier application-level checkpoint/restart and automatic fault tolerance," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 3, pp. 501-514, 2019, doi: 10.1109/TPDS.2018.2866794.

- [46] F. Li et al., "Checkpointing-aware loop tiling for energy harvesting powered nonvolatile processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 1, pp. 15-28, 2019, doi: 10.1109/TCAD.2018.2803624.
- [47] K. Dichev, D. De Sensi, D.S. Nikolopoulos, K.W. Cameron, and I. Spence, "Power Log'n'Roll: Power-efficient localized rollback for MPI applications using message logging protocols," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1276-1288, 2022, doi: 10.1109/TPDS.2021.3107745.
- [48] P. Sigdel, X. Yuan, and N.-F. Tzeng, "Realizing best checkpointing control in computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 315-329, 2021, doi: 10.1109/TPDS.2020.3015805.
- [49] B. Schroeder and G.A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337-350, 2010, doi: 10.1109/TDSC.2009.4.
- [50] M. Chtepen et al., "Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 180-190, 2009, doi: 10.1109/TPDS.2008.93.
- [51] E. Deelman et al., "Pegasus, a workflow management system for science automation," *Future Gener. Comput. Syst.*, vol. 46, pp. 17-35, 2015, doi: 10.1016/j.future.2014.10.008.
- [52] M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Comput.*, vol. 37, no. 9, pp. 633-652, 2011, doi: 10.1016/j.parco.2011.05.005.
- [53] L. Han, L.-C. Canon, H. Casanova, Y. Robert, and F. Vivien, "Checkpointing workflows for fail-stop errors," *IEEE Trans. Comput.*, vol. 67, no. 8, pp. 1105-1120, 2018, doi: 10.1109/CLUSTER.2017.14.
- [54] A. Abdi, S.A. Asghari, H. Pedram, and H. Taheri, "An instruction level software redundancy based method to intra-inter block control flow checking," *Soft Comput. J.*, vol. 1, no. 1, pp. 56-69, 2012, doi: 20.1001.1.23223707.1391.1.1.114.1 [In Persian].
- [55] E. Asadollahi and S. A. Asghari, "Prediction of appropriate number of virtual machines based on time series and artificial methods via virtual machines clustering," *Soft Comput. J.*, vol. 6, no. 1, pp. 66-77, 2017 [In Persian].
- [56] M.J. Nadjafi-Arani and S. Doostali, "Cost-based workflow scheduling using algebraic structures," *Soft Comput. J.*, vol. 9, no. 2, pp. 114-129, 2021, doi: 10.22052/scj.2021.242814.0. [In Persian]
- [57] M. Salehi et al., "Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 7, pp. 2426-2437, 2016, doi: 10.1109/TVLSI.2015.2512839.
- [58] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," *Real-Time Syst.*, vol. 20, no. 1, pp. 83-102, 2001, doi: 10.1023/A:1026589200419.
- [59] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 217-231, 2004, doi: 10.1109/TC.2004.1261830.
- [60] K.H. Kim and J. Kim, "An adaptive DVS checkpointing scheme for fixed-priority tasks with reliability constraints in dependable real-time embedded systems," in *Proc. 3rd Int. Conf. Embedded Softw. Syst. (ICCESS)*, 2007, pp. 560-571, doi: 10.1007/978-3-540-72685-2\_52.
- [61] Y. Li and Z. Lan, "FREM: A fast restart mechanism for general checkpoint/restart," *IEEE Trans. Comput.*, vol. 60, no. 5, pp. 639-652, 2011, doi: 10.1109/TC.2010.129.