مروری بر روش‌های کاهش سربار نقطه‌وارسی

مصطفی آغبلاغی تبریزی[1*]، دانشجو کارشناسی ارشد، آرین شهبازیان[2]، دانشجو کارشناسی ارشد، کیمیا شاه‌بختی[3]، دانشجو کارشناسی ارشد، کوشیار زمانی[4]، دانشجو کارشناسی ارشد، نوید کنعانی[5]، دانشجو کارشناسی ارشد، سید امیر اصغری[6*]، دانشیار، محمدرضا بینش مروستی[7]، دانشیار

[1] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، m.aghbolaghitabrizi@khu.ac.ir

[2] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، arian.shahbazian@khu.ac.ir

[3] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، kimiyashahbakhti@khu.ac.ir

[4] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، kooshyar.zamani@khu.ac.ir

[5] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، navidkanaani@khu.ac.ir

[6] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، asghari@khu.ac.ir

[7] گروه مهندسی برق و کامپیوتر، دانشکده فنی مهندسی، دانشگاه خوارزمی، تهران، ایران، marvasti@khu.ac.ir

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 آنهاست. 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 وارسی هماهنگ 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.

واژه‌های کلیدی: نقطه‌وارسی، کاهش سربار نقطه‌وارسی، نقطه‌وارسی هماهنگ شده، نقطه‌وارسی در سطح سیستم، نقطه‌وارسی در سطح برنامه، نقطه‌وارسی در سیستم‌های محاسباتی توزیع شده

* سید امیر اصغری، asghari@khu.ac.ir – مصطفی آغبلاغی تبریزی، m.aghbolaghitabrizi@khu.ac.ir

A Survey of Checkpoint Overhead Reduction Methods

Mostafa Aghbolaghi Tabrizi [1*], Master Student, Arian Shahbazian [2], Master Student, Kimiya Shahbakhti [3], Master Student, Kooshyar Zamani [4], Master Student, Navid Kanaani[5], Master Student, Seyyed Amir Asghari[6*], Associative Professor, Mohammad Reza Binesh Marvasti[7], Associative Professor

[1] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, m.aghbolaghitabrizi@khu.ac.ir

[2] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, arian.shahbazian@khu.ac.ir

[3] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, kimiyashahbakhti@khu.ac.ir

[4] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, kooshyar.zamani@khu.ac.ir

[5] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, navidkanaani@khu.ac.ir

[6] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, asghari@khu.ac.ir

[7] Dept. of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran, marvasti@khu.ac.ir

Abstract: Nowadays, fault tolerance in different systems is a very essential factor. Using checkpointing methods and safe spots for recovery after failure occur can increase the reliability of systems. The main issue with using checkpointing methods is their overhead. This overhead made as a result of checkpointing execution and it has negative impact on system performance. Therefore, numerous methods have been introduced to address this problem. These methods aim to reduce the overhead in order to increase system performance. This paper, thoroughly studied and reviewed various checkpointing methods. These methods organized into distinct categories. Then categories are defined based on the type of checkpoint implementation and different levels of systems. Those are such as: coordinated checkpointing, system-level checkpointing, application-level checkpointing, and distributed system checkpointing. Finally, this paper provides a detailed summary in a comprehensive graph and conclusion for each of these categories.

Keywords: *Checkpoint; Checkpoint Overhead Reduction; Coordinated Checkpoint; System-level Checkpoint; Application-level Checkpoint; Checkpoint in distributed computing systems*

*** Seyyed Amir Asghari, asghari@khu.ac.ir - Mostafa Aghbolaghi Tabrizi, m.aghbolaghitabrizi@khu.ac.ir**

۱. مقدمه

نقطه‌وارسی[1] روشی پرکاربرد برای افزایش تحمل‌پذیری اشکال در سیستم‌های کامپیوتری است [1]. ازسرگیری دوباره فعالیت سیستم پس از مواجه با خطا با استفاده از اطلاعات ذخیره شده در نقطه‌وارسی، باعث افزایش کارایی سیستم می‌شود. نقطه‌وارسی با وجود تحقق اهدافی مانند کاهش اتلاف زمان در هنگام مواجه با خرابی و شکست، سربار قابل‌توجهی 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 روبه‌روست. این سربار به دلیل توقف اجرای برنامه اصلی سیستم به هنگام ایجاد نقطه‌وارسی ایجاد می‌شود. این وقفه 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 نقطه‌وارسی عملیات نوشتن در حافظه پایدار است. یک حافظه پایدار مکانی برای ذخیره نقطه‌وارسی است تا در صورت بروز اشکال بتوان حالت سیستم را از این حافظه بازیابی کرد. نوشتن نقطه‌وارسی در حافظه پایدار باعث ایجاد شلوغی در ورودی‌های خواندن و نوشتن حافظه می‌شود. این شلوغی در نهایت سبب تأخیر دسترسی به حافظه پایدار می‌شود و همچنین دیگر برنامه‌ها نیز تحت‌تأثیر این تأخیر قرار می‌گیرند [4].

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

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

برخی روش‌ها با کاهش تعداد نقطه‌وارسی باتوجه‌به شرایط سیستم، در تلاش‌اند کارایی کلی سیستم را افزایش دهند. با کاهش تعداد نقطه‌وارسی زمان توقف سیستم برای انجام فرآیند نقطه‌وارسی کاهش می‌یابد. این بهبود باید به‌گونه‌ای 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 نقطه‌وارسی کرد. فرآیند ایجاد نقطه‌وارسی در پژوهش‌های چند سال اخیر موردتوجه بسیار قرار گرفته است. اینکه از چه اطلاعاتی برای ایجاد نسخه پشتیبان استفاده شود و یا چه بخش‌هایی در طول زمان اجرای فرآیند نقطه‌وارسی صرف کدام فعالیت‌ها شود، از زمینه‌های مورد بررسی محققان بوده است.

[1] Checkpoint

به دلیل گسترده بودن کاربرد نقطه‌وارسی، کاهش سربار نقطه‌وارسی می‌تواند در سطوح مختلفی انجام شود. نقطه‌وارسی می‌تواند شامل حالت پردازنده باشد. حالت پردازنده مقادیر ثبات‌های فرار است که طی فرآیند نقطه‌وارسی 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 دست‌یافت.

در گذشته مقاله‌های مروری مختلفی در ارتباط با نقطه‌وارسی و مفاهیم پایه‌ای آن منتشر شده است [8]. مبحث نقطه‌وارسی با تمام سادگی در مفهوم، در جزئیات بسیار پر محتوا و قابل‌بحث است. بررسی‌های مختلف در مفاهیم نقطه‌وارسی این نتیجه را داشته است که توجه بسیاری از مقالات، پیرامون کاهش سربار نقطه‌وارسی 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 نقطه‌وارسی شده است. در این مقاله معیار مقایسه جدیدی معرفی نشده است و فقط به‌مرور ادبیات پرداخته شده است. روش‌های کاهش سربار نقطه‌وارسی در دسته‌بندی‌های مختلفی معرفی شده‌اند. هر روش به‌صورت 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 گزارش‌های نویسندگان و ادعای آن‌ها بیان شده است. معتزالنزهی و همکاران در مروری کلی روش‌های مبتنی بر بازگشت - بازیابی را معرفی کرده‌اند [9]. آن‌ها در مطالعه‌های خود نگاهی کلی به سربار ناشی از نقطه‌وارسی داشته‌اند.

۲. 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 شده اند، 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 معرفی‌شده و سپس در ادامه روش‌های مرتبط به آن دسته‌بندی بررسی شده است و در بخش ۳، معیارهای بررسی شرح داده شده است.

۱.۲. نقطه‌وارسی هماهنگ شده[1]

در نقطه‌های وارسی هماهنگ شده نیاز است تا همه فرآیندها به‌منظور 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[2] 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 حالت سیستم سازگار باقی بماند. در نقطه‌وارسی هماهنگ شده، فرآیندها با همگام‌سازی فرآیند ایجاد نقطه‌های

[1] Coordinated Checkpointing
[2] Domino effect

شکل (1). روش‌های کاهش سربار نقطه‌وارسی

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 هماهنگ‌شده

۱.۱.۲. نقطه‌وارسی چند سطحی بدون دیسک[1]

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[2] استفاده نمی‌کند و مشکل گلوگاه و تأخیر در پاسخ‌دهی ندارد [13].

روش ارائه شده توسط Hakkarinen و همکاران که نقطه‌وارسی چندسطحی بدون دیسک نام دارد، کاهش سربار 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 وارسی 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 هماهنگ‌کردن این نقطه‌های وارسی 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 مدیریت‌کردن 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 تأخیر ناشی از پهنای باند را از بین می‌برد و می‌تواند کارایی بالاتری ارائه کند.

۲.۱.۲. روش نقطه‌وارسی آنلاین دوسطحی بهبود یافته

در سیستم‌های محاسباتی با کارایی بالا[3] در صورت بروز هر نوع خرابی اگر سیستم راه‌اندازی دوباره شود و محاسبات قبلی از دست برود، علاوه بر اتلاف منابع با کاهش قابل‌توجه کارایی مواجه هستیم. روش نقطه‌وارسی چندسطحی[4] برای رفع مشکل سربار نقطه‌وارسی معرفی شده است [14] - [16].

در روش نقطه‌وارسی دوسطحی در سطح اول با خطاهای گذرا که سربار نقطه‌وارسی کمی دارند (مانند خطاهای حافظه) سروکار داریم. سطح دوم بر خرابی‌های سخت‌افزاری مانند

[1] Multilevel Diskless Checkpointing
[2] Stable Storage
[3] **H**igh-**P**erformance **C**omputing (HPC)
[4] Multilevel Checkpointing

خرابی نودها یا گره‌ها تمرکز دارد. در این مقاله [17]، نویسندگان در ابتدا با فرموله‌کردن شرایط آنلاین و آفلاین به نتایجی دست می‌یابند که منجر به رسیدن یک روش بهینه برای نقطه‌وارسی چندسطحی می‌شود. در بخش اول با فرض نقطه‌وارسی آفلاین و دانستن طول کلی الگو پیش از شروع اجرا، اقدام به اثبات رابطه زمان موردنیاز برای اجرای الگو می‌کنند. باتوجه 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 نشان‌داده‌شده 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 به‌دست‌آمده 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.

۳.۱.۲. روش مبتنی بر گراف در سیستم‌های موبایل

از دیگر زمینه‌های استفاده از نقطه‌وارسی، سیستم‌های عامل سیار[1] 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 این‌گونه 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 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 تأییدکننده 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 تأخیر زمانی روش وارسی جهانی را به میزان قابل‌توجهی کاهش می‌دهد. ازآنجاکه 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.

۴.۱.۲. 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[2] روشی است که اخیراً پیشنهاد شده است [20].

این روش با درنظرگرفتن ویژگی دمای پردازنده سعی می‌کند تا ماژول‌های نزدیک به خرابی را پیش‌بینی کند. بعد از پیش‌بینی

[1] Mobile Agent

[2] **P**roactive **C**oordinated **F**ault-**T**olerance (PCFT)

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[1] 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.

۵.۱.۲. ایجاد یا عدم ایجاد نقطه‌وارسی ، تقابل مصرف انرژی و سربار

در سیستم‌های محاسباتی با کارایی بالا با مقیاس بزرگ، تعادل میان دو فاکتور مصرف انرژی و قابلیت اطمینان یک چالش است. ایده فراهم‌کردن 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 به‌صورت پویا فواصل نقطه‌وارسی را بر اساس الگوهای خرابی تنظیم می‌کنند تا مصرف انرژی را بهینه کنند. این روش با درنظرگرفتن 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 صرفه جویی 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.

۶.۱.۲. انتخاب بهینه نقطه‌وارسی با افزونگی دو مدولار

امروزه با افزایش پیچیدگی سیستم‌های روی تراشه نرخ خرابی افزایش زیادی پیدا کرده‌است. ازاین‌رو بایستی روش‌هایی را برای تضمین قابلیت اطمینان این‌گونه سیستم‌ها بررسی کرد. تحمل اشکالات گذرا با انتخاب موقعیت‌های بهینه نقطه‌وارسی به‌منظور به‌حداقل‌رساندن سربار هدف پژوهش Kang و همکاران بوده است [22]. آن‌ها با استفاده از افزونگی دو مدولار[2] 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 در مرز اجرای وظیفه[3] محاسبه می‌شود. با استفاده از معرفی فاکتور بودجه زمانی[4] سعی در یافتن نقطه‌وارسی بهینه می‌کند 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 صورت‌گرفته روش پیشنهادی به طور متوسط ۶۸/۶ برابر برای تک پردازنده‌ها و ۸۶/۲ برابر برای چندپردازنده‌ها کاهش سربار نقطه‌وارسی را به همراه داشته است.

۷.۱.۲. نقطه‌وارسی HeteroCheckpoint کارآمد برای شتاب‌دهنده‌ها

افزایش استفاده از واحدهای پردازش گرافیکی[5] در سیستم‌های محاسباتی با کارایی بالا منجر به نرخ خرابی بالاتر در مقایسه با

[1] **P**article **S**warm **O**ptimization (PSO)

[2] **D**ual **M**odular **R**edundancy (DMR)

[3] Task Execution Boundary

[4] Time Budget

[5] GPU

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].

پیش کپی داده‌ها در سطح تکه[1] می‌تواند به طور مؤثر زمان اجرای کلی برنامه را کاهش دهد. این عملیات پیش کپی 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 قابل‌توجهی در سربار نقطه‌وارسی می‌شود. روش دیگر استفاده از نشان‌گذاری علامت کثیف برای متغیرهای تغییریافته در زمان اجرا است. این نشان‌گذاری پویا توسط واحد پردازش گرافیکی انجام می‌شود و باعث می‌شود هزینه کنترلی متغیرها برای ایجاد نقطه‌وارسی کاهش یابد. این روش که بهبودی بر روش‌های پیشین است توانسته سربار نقطه‌وارسی و راه‌اندازی دوباره را تا ۶۰٪ نسبت به روش‌های سنتی کاهش دهد. این روش علی‌رغم بهبود در سربار نقطه‌وارسی باعث افزایش استفاده از حافظه پردازنده می‌شود که نیازمند بررسی است.

۸.۱.۲ SCR کتابخانه‌ای مقیاس پذیر برای نقطه‌وارسی

در سیستم‌های محاسباتی با کارایی بالا که شاهد پیشرفت قابل‌توجهی 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 گرفته‌شده 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 وارسی 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 قابل‌توجه 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 وارسی ذخیره شده در کش بازیابی را انجام داد.

۲.۲. 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

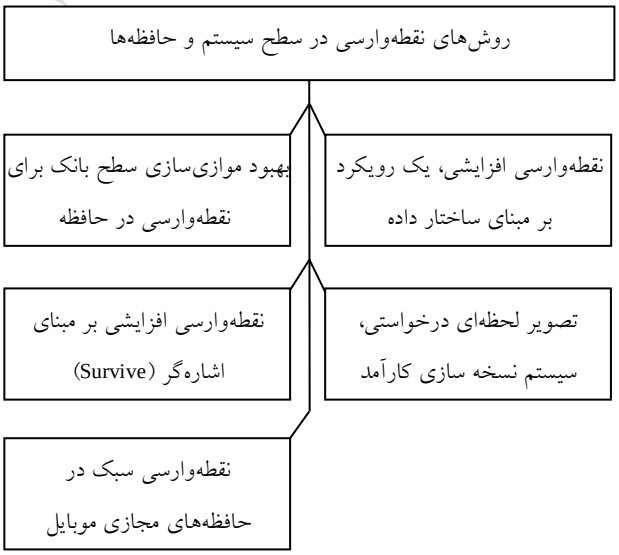
[1] Chunk

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 به‌دور 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

۱.۲.۲. بهبود موازی سازی سطح بانک برای نقطه‌وارسی در حافظه

نقطه‌وارسی در سطح بانک‌های حافظه می‌تواند باعث تداخل در 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 این 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.

۲.۲.۲. تصویر لحظه‌ای درخواستی، یک سیستم نسخه سازی کارآمد

یکی دیگر از کاربردهای نقطه‌وارسی استفاده در حافظه‌های فاز متغیر[1] است. در روش تصویر لحظه‌ای درخواستی[2] نویسندگان با کاهش سربار نوشتن و به‌حداقل‌رساندن نوشتن‌های بیهوده به هنگام تهیه تصویر لحظه‌ای، سعی در افزایش کارایی داشته‌اند [33].

[1] **P**hase-**C**hange **M**emory (PCM)
[2] On-Demand Snapshot

آن‌ها در روش خود به‌جای نوشتن تمام فایل، تنها بخش‌های تغییریافته 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 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 باتوجه 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 به‌صورت 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.

۳.۲.۲. نقطه‌وارسی افزایشی، یک رویکرد بر مبنای ساختار داده

با افزایش تقاضا برای پردازش با سرعت بالا، پایگاه‌های داده داخل حافظه به طور گسترده‌ای برای تجزیه‌وتحلیل 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[1] 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 کرده اند. 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 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.

۴.۲.۲. 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[2] معرفی شده است [39].

در این روش صفحات حافظه مجازی دسته‌بندی می‌شوند. این دسته‌بندی‌ها شامل حالات ماشین مجازی، مجموعه صفحات

[1] **D**ata **S**tructure based **I**ncremental **C**heckpointing (DSIC)

[2] **F**ast and **Li**ghtweight **C**heckpointing (FLIC)

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 تاحدامکان 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.

۵.۲.۲. Survive، روش نقطه‌وارسی افزایشی بر مبنای اشاره‌گر

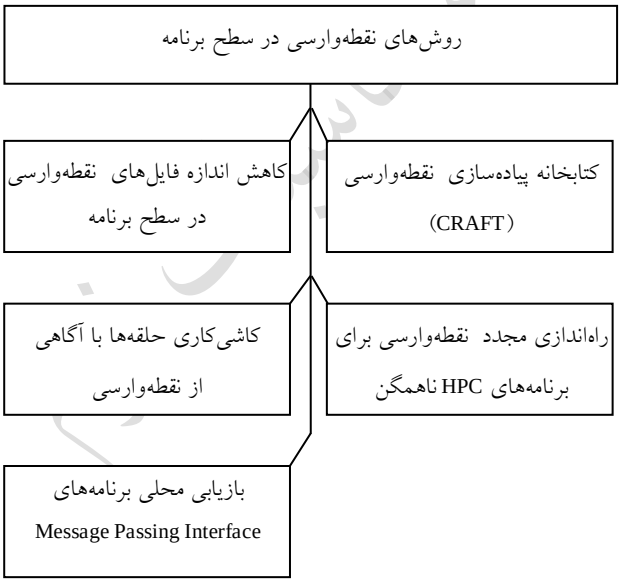
در این مقاله [40] نویسندگان روشی بهبودیافته برای نقطه‌وارسی در سطح حافظه‌های پویا معرفی کرده‌اند. اساس این روش بر مبنای نقطه‌وارسی افزایشی است [41]. در مواقع قطعی برق به دلیل فرار بودن اطلاعات ذخیره‌شده نقطه‌وارسی امکان از سرگیری دوباره فعالیت‌ها وجود ندارد.

نویسندگان در روش خود که آن را Survive نامیده‌اند، با استفاده از ذخیره آخرین حالت در یک حافظه غیرفرار این امکان را فراهم کرده‌اند که در صورت قطعی برق هم بتوان دوباره اجرا را با وصل‌شدن 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 غیرفرار ذخیره می‌گردد. حافظه غیرفرار در صورت مشغول نبودن حافظه توسط دستورات خواندن و نوشتن، به‌روزرسانی می‌شود. با منتقل‌شدن ردیف‌های لیست نقطه‌وارسی به حافظه غیرفرار می‌توان ورودی‌ها را از لیست پاک کرد.

این روش نسبت به روش غیربهینه تنها ٪3/5 سربار زمان اجرا تحمیل می‌کند اما در مقابل، سازگاری در برابر خرابی را فراهم می‌کند.

۳.۲. 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 به‌آسانی 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 نشان‌داده‌شده است.



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

۱.۳.۲. 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 این‌گونه برنامه‌ها روش‌های سنتی نقطه‌وارسی باعث ایجاد فایل‌های حجیمی می‌شوند [42].

هدف نویسندگان این 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 شده‌است. 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 این‌گونه 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.

۲.۳.۲. 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 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 خرابی‌هاست. 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 به‌صورت ساده یک شیء از نقطه‌وارسی ساخته می‌شود و تمام داده‌های مرتبط به آن توسط تابع افزودن اضافه می‌شود. زمانی که تمام داده‌های مرتبط اضافه شد، شیء 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.

۴.۳.۲. کاشی‌کاری حلقه‌ها با آگاهی از نقطه‌وارسی

در این مقاله [46] به سربار نقطه‌وارسی در اجرای حلقه‌های تودرتو پرداخته شده‌است. 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 بدست 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.

در روش کاشی‌کاری[1] با تغییر کد حلقه به حالتی که اجرا به صورت کاشی طور باشد از وابستگی داده‌ای چند نود در هر کاشی جلوگیری می‌شود که در نتیجه نیازی به ایجاد نقطه‌وارسی

[1] Tiling

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.

۵.۳.۲. روش بازیابی محلی برای برنامه‌های رابط فرستادن پیام

برای استفاده از خوشه‌های متشکل از چند کامپیوتر مشخصه رابط فرستادن پیام[1] در برنامه‌ها کاربرد دارد. در تکنیک‌های مبتنی بر بازگشت محلی با افزایش اجزا و ابزارهای محاسباتی، تأثیر منفی در واحد متوسط زمان میان خرابی‌ها مشاهده شده است.

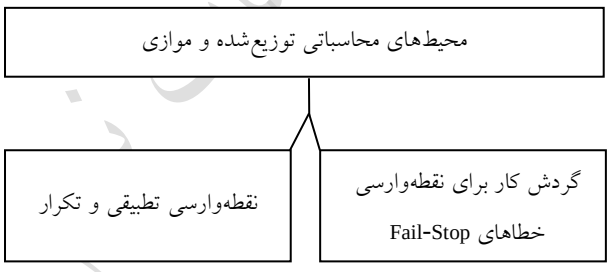
کار تحقیقی گروه Kiril Dichev برای کاهش سربار در این‌گونه روش‌ها و همچنین بهبود مصرف انرژی در این‌گونه سیستم‌هاست [47]. نویسندگان با ارائه الگوریتمی تنها به کاهش هزینه‌های نقطه‌وارسی در فرآیندهای فعال و خراب پرداخته‌اند. دلیل این کار این است که تنها این فرآیندها در پروسه بازیابی دخیل هستند. در عملیات 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[2] است که محدودیت کاربردی روش‌های پیشین را 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.

۴.۲. روش‌های نقطه‌وارسی در محیط‌های محاسباتی توزیع شده

در محیط‌های محاسباتی توزیع شده استفاده از نقطه‌وارسی یک امر مهم است. زیرا این رویکرد باعث می‌شود سیستم قابلیت تحمل‌پذیری اشکال پیدا کند. به دلیل اجرای طولانی‌مدت نرم‌افزارها در این‌گونه محیط‌ها، ایجاد یک تصویر کلی از حالت نرم‌افزار در لحظه ایجاد نقطه‌وارسی امری ضروری است. این نقطه‌وارسی باعث می‌شود تا در صورت بروز اشکال در سیستم، نرم‌افزار بتواند از آخرین نقطه‌وارسی 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 طولانی‌بودن مدت‌زمان اجرای برنامه‌هایشان هستند.

روش‌های کاهش سربار نقطه‌وارسی در محیط‌های محاسباتی توزیع‌شده و موازی در شکل 5 نمایش‌داده‌شده است.



شکل (5). روش‌های کاهش سربار نقطه‌وارسی در محیط‌های محاسباتی توزیع‌شده و موازی

[1] **M**essage **P**assing **I**nterface (MPI)

[2] Message Logging

۱.۴.۲. نقطه‌وارسی 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 نقطه‌وارسی در این سیستم‌ها هدفی بود که باعث پیشنهاد الگوریتم‌های نقطه‌وارسی وابسته به آخرین شکست[1] و نقطه‌وارسی وابسته به میانگین شکست[2] شد [50].

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

۲.۴.۲. گردش کار برای نقطه‌وارسی خطاهای توقف/شکست

در سیستم‌های محاسبات موازی، روش‌های پیشین Checkpoint-All 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 نقطه‌وارسی داشته‌اند. مقاله [53] با استفاده از ارتباط و وابستگی بین فرآیندها سعی در بهبود سربار دارد. این روش Checkpoint-Some نام دارد.

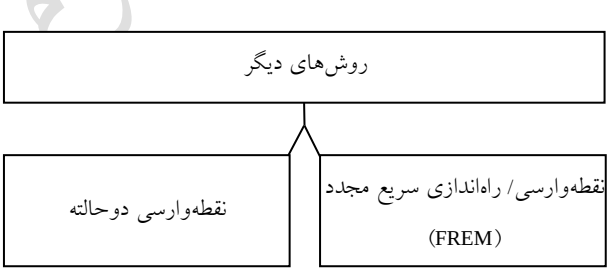
در این روش با استفاده از سری گراف‌های موازی کمینه، امکان محاسبه مجموعه‌ای از کارهای مستقل در یک پردازنده و ارتباط با پردازنده‌های دیگر را فراهم می‌سازد. با ایجاد نقطه‌وارسی از کارهای خروجی یک Superchain وابستگی‌های متقاطع از بین می‌رود. این روش یک زمان‌بندی را برای ایجاد نقطه‌وارسی معرفی می‌کند تا میان داده‌های نقطه‌وارسی وابستگی وجود نداشته باشد. با ایجاد این‌گونه نقطه‌های وارسی 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 قابل‌توجه در سربار نقطه‌وارسی شده است.

۵.۲. روش‌های دیگر

در این بخش چند روش کاهش سربار 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 سربار نقطه‌وارسی

[1] Last Failure Dependent Checkpointing
[2] Mean Failure Dependent Checkpointing

۱.۵.۲. نقطه‌وارسی دوحالته برای تحمل‌پذیری اشکال و بهره‌وری انرژی

استفاده از نقطه‌های وارسی 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 وارسی در دوره بدون خطا سیستم می‌تواند با کاهش میانگین تعداد نقطه‌های وارسی 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 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 وارسی را کاهش می‌دهد. سربار کاهش یافته با این روش نسبت به روش‌های گذشته ٪14 اعلام شده است. سایر روش‌هایی که مورد مقایسه با روش بالا قرارگرفته‌اند، شامل نقطه‌وارسی منظم بهینه[1] [58]، نقطه‌وارسی نامنظم[2] [59] و نقطه‌وارسی منظم انطباقی[3] [60] هستند.

2.5.2. FREM، روش کلی راه‌اندازی مجدد سریع برای نقطه‌وارسی/ راه‌اندازی مجدد

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

روش FREM با همپوشانی بازیابی برنامه با تصویر نقطه‌وارسی آن، تأخیر 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 به‌صورت هم‌پوشانی اجرا می‌شوند تا درصورتی‌که در پنجره ردیابی اشکالی رخ دهد، فرآیند ازسرگرفته شود. در بخش FR هدف FREM دستیابی به همپوشانی بین اجرای فرآیند و بازیابی نقطه‌وارسی است. در نتیجه تأخیر راه‌اندازی مجدد به زمان بازیابی با اطلاعات نقطه‌وارسی را کاهش می‌دهد. روش پیشنهادی این مقاله ویژگی‌های کارایی، عمومیت و پنهان بودن از دید کاربر را دارد. از طرفی به دلیل انجام عملیات رصدکردن و بازیابی سریع، مقداری سربار ایجاد می‌کند که طبق گفته نویسندگان، به دلیل مزایای این روش می‌توان از آن چشم‌پوشی شود.

۳. نتایج و پیشنهادهای آتی

کاهش سربار نقطه‌وارسی هدف مشترک همه روش‌های بررسی شده بوده است. در نقطه‌های وارسی هماهنگ شده برای کاهش سربار بایستی اصل سازگاری سیستم را هم رعایت کرد.

[1] Optimal Uniform Checkpointing
[2] NonUniform Checkpointing
[3] Adaptive Uniform Checkpointing

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 پیش آید؛ 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 بیهوده 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 قابل‌پیاده‌سازی است. باتوجه به اطلاعات و داده‌های موجود در سیستم‌ها برای ویژگی‌های ذکر 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 به‌حداقل‌رساندن اطلاعات هر نقطه‌وارسی داشته‌اند تا بتوانند به کاهش سربار نقطه‌وارسی کمک کنند. برای مثال روش نقطه‌وارسی افزایشی باتوجه‌به 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 قابل‌تحقق 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 وارسی 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 شده‌است.

جدول (1). مقایسه روش‌های کاهش سربار نقطه‌وارسی

نام روش	مزایا	معایب	سال انتشار	شماره منبع
بهبود موازی‌سازی سطح بانک برای نقطه‌وارسی در حافظه	بهبود عملکرد با کاهش تداخل نقطه‌وارسی و برنامه‌های در حال اجرا	• طراحی مجدد کنترل‌کننده حافظه • نیاز به پیاده‌سازی بخش‌های جدید سخت‌افزاری	2022	[32]
روش بازیابی محلی برای برنامه‌های رابط فرستادن پیام	• بهبود مصرف انرژی • قابل‌اعمال بر روی اکثر سیستم‌های محاسباتی با کارایی بالا	• نیاز به ابزاردقیق برای کدنویسی • فقط تماس‌های نقطه‌به‌نقطه در نمونه اولیه ثبت شده‌اند	2022	[47]
ایجاد یا عدم ایجاد نقطه‌وارسی	• صرفه‌جویی در مصرف انرژی با تأثیر کمتر در کارایی • قابلیت تطبیق	• پیچیدگی سیستم نقطه‌وارسی به دلیل پیاده‌سازی و مدیریت • سیاست‌های تطبیق	2014	[21]
HeteroCheckpoint نقطه‌وارسی کارآمد برای شتاب‌دهنده‌ها	• کاهش سربار نقطه‌وارسی تا ۶۰ درصد نسبت به روش‌های سنتی • بهبود عملکرد • کاهش پهنای باند انتها به انتها	• افزایش استفاده از حافظه پردازنده • سربار مربوط به محاسبات چک‌سام	1992	[24]
کاهش اندازه فایل‌های نقطه‌وارسی در سطح برنامه	• کاهش قابل‌توجه اندازه نقطه‌وارسی • حداقل سربار عملکرد در مقایسه با نقطه‌وارسی فشرده نشده (کمتر از ۲ درصد) • مقیاس‌پذیری در سیستم‌های بزرگ	پیچیدگی بیشتر نسبت به نقطه‌های وارسی ساده	2012	[43]
راه‌اندازی مجدد نقطه‌وارسی برای برنامه‌های HPC ناهمگن	• کاهش زمان در نقطه‌وارسی و بازیابی (تا ۱۵ برابر بهبود) • وجود نقطه‌وارسی دیفرانسیلی و کاهش هزینه تا ۲,۶ درصد • پشتیبانی از معماری‌های متنوع و پیچیده • بهبود عملکرد کلی	افزایش پیچیدگی به دلیل اضافه‌شدن مدیریت حافظه و تکنیک‌های بهینه‌سازی	2012	[44]
استفاده از روش فعال نقطه‌وارسی	• بهبود زمان کلی اجرا • کاهش خرابی منجر به بازیابی	پارامترهای دریافتی در این روش بایستی قابل‌اتکا باشند	2018	[20]
نقطه‌وارسی دوحالته برای تحمل‌پذیری اشکال و بهره‌وری انرژی	• افزایش کارایی کلی سیستم با کاهش تعداد نقطه‌های وارسی • کاهش مصرف انرژی	زمان مواجه سیستم با خطا و تغییر نوع نقطه‌وارسی همچنان شاهد کاهش عملکرد هستیم	2016	[57]
نقطه‌وارسی سبک در حافظه‌های مجازی موبایل	• کاهش اطلاعات ذخیره شده در هر نقطه‌وارسی • بهبود مصرف انرژی	در ایجاد نقطه‌وارسی از صفحات بیکار حافظه کند عمل می‌کند	2019	[39]
روش نقطه‌وارسی آنلاین دوسطحی بهبودیافته	• بهبود زمان کلی اجرا • تحمل‌پذیری اشکال در سطوح مختلف	در روش آنلاین پیشنهاد شده، بهبودها مانند روش آفلاین است	2017	[17]
Survive، روش نقطه‌وارسی افزایشی بر مبنای اشاره‌گر	• افزایش قابلیت اطمینان سیستم • افزایش اولویت دیگر برنامه‌ها به حافظه غیرفرار	ذخیره اشاره‌گرها سربار حافظه دارد	2017	[40]

نام روش	مزایا	معایب	سال انتشار	شماره 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	باتوجه‌به 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 وارسی	• 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 وارسی در هر گره	2010	[27]
نقطه‌وارسی تطبیقی و تکرار، روشی کارآمد برای تحمل اشکال	گرفتن نقطه‌وارسی باتوجه‌به پارامترهای مختلف که باعث کاهش گرفتن نقاط وارسی بیهوده 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	درصورتی‌که تعداد دستورات ما بین دو نقطه‌وارسی کمتر از 100 دستور باشد، بهبود عملکرد و کاهش سربار دارد	وابسته به تعداد دستورات افزودن/حذف‌کردن در میان نقطه‌های وارسی است	2017	[35]

4. نتیجه‌گیری

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

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 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 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 وارسی، 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 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.

5. مراجع

[1] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A Survey of Fault Tolerance Mechanisms and Checkpoint/Restart Implementations for High Performance Computing Systems", *The Journal of Supercomputing*, vol. 65, pp. 1302-1326, Feb 2013, doi: 10.1007/s11227-013-0884-0

[2] N. H. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme," *in IEEE Transactions on Computers*, vol. 46, no. 8, pp. 942-947, Aug. 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," in IEEE Transactions on Parallel and Distributed Systems, vol. 35, no. 7, pp. 1307-1319, July 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," 2016 International Wireless Communications and Mobile Computing Conference (IWCMC), Paphos, Cyprus, 2016, pp. 672-677. doi: 10.1109/IWCMC.2016.7577137

[5] Kai Hwang, Hai Jin, Edward Chow, Cho-Li Wang, and Zhiwei Xu, "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space", *IEEE concurrency*, 7(1):60–69, 1999. doi: 10.1109/4434.749136

[6] Xiaole Cui, Zuolin Cheng, Chunglen Lee, Xinnan Lin, Yiqun Wei, Xiaogang Chen, and Zhitang Song, " A Snake Addressing Scheme for Phase Change Memory Testing" *Science China Information Sciences*, 59(10):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, June 2003. Doi: 10.1145/966049.781513

[8] Egwutuoha, I.P., Levy, D., Selic, B. et al. "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems." *J Supercomput* 65, 1302–1326, 2013. doi: 10.1007/s11227-013-0884-0

Approach to Enhance Cloud Service Reliability," in *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191-1202, 1 Oct.-Dec. 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," *IEEE International Conference on Cluster Computing (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," in *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 2036-2048, 1 July 2015.
doi: 10.1109/TC.2014.2349492

[23] K. B. Ferreira, R. Riesen, R. Brighwell, P. Bridges, and D. Arnold, "Recent Advances in the Message Passing Interface" in *Springer* 2011.
doi: 10.1177/1094342014549273

[24] S. Kannan, N. Farooqui, A. Gavrilovska and K. Schwan, "HeteroCheckpoint: Efficient Checkpointing for Accelerator-Based Systems," *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 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 *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008, pp. 153–162.
doi: 10.1145/1345206.1345230

[26] E. Gelenbe, "A Model of Roll-back Recovery with Multiple Check- points," in *Proceedings of the 2nd International Conference on Software Engineering* (ICSE '76), 1976, pp. 251–255.
doi: 10.5555/800253.807684

[27] A. Moody, G. Bronevetsky, K. Mohror and B. R. d. Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," SC '10: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing*, Networking, Storage and Analysis, New Orleans, LA, USA, 2010, pp. 1-11.
doi: 10.1109/SC.2010.18

[28] X. Tang, J. Zhai, B. Yu, W. Chen, W. Zheng and K. Li, "An Efficient In-Memory Checkpoint Method and its Practice on Fault-Tolerant HPL," in IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 4, pp. 758-771, 1 April 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," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 9, pp. 1497-1510, Sept. 2017.
doi: 10.1109/TCAD.2017.2651824

[9] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. 2002. "A survey of rollback-recovery protocols in message-passing systems." *ACM Comput. Surv.* 34, 3 (September 2002), 375–408.
doi: 10.1145/568522.568525

[10] Guohong Cao and M. Singhal, "On coordinated checkpointing in distributed systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 12, pp. 1213-1225, Dec. 1998.
doi: 10.1109/71.737697

[11] M. Taqi Raza, Z. Tan, A. Tufail and F. Muhammad Anwar, "LTE NFV Rollback Recovery," in IEEE Transactions on Network and Service Management, vol. 19, no. 3, pp. 2468-2477, Sept. 2022.
doi: 10.1109/TNSM.2022.3182008

[12] J. Plank, K. Li, and M. Puening, "Diskless Checkpointing," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 10, pp. 972-986, Oct. 1998.
doi: 10.1109/71.730527

[13] D. Hakkarinen and Z. Chen, "Multilevel Diskless Checkpointing," in *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 772- 783, April 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.*, pp. 1–10, 2010.
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," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 244-259, 1 Jan. 2017.
doi: 10.1109/TPDS.2016.2546248

[18] S.S. Manvi and P. Venkataram, "Applications of Agent Technology in Communications: A Review," *Springer Computer Comm.*, 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," in *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 247-258, Feb. 2013.
doi: 10.1109/TC.2011.235

[20] J. Liu, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang and R. Buyya, "Using Proactive Fault-Tolerance

Architecture Letters, vol. 16, no. 2, pp. 153-157, 1 July-Dec. 2017.
doi: 10.1109/LCA.2016.2646340

[41] M. Prvulovic, Z. Zhang, and J. Torrellas, "ReVive: Cost-effective architec- tural support for rollback recovery in shared-memory multiprocessors," in *Proc. 29th Annu. Int. Symp. Comput. Archit*., 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". *International Journal of High Performance Computing Applications* 23(3), 212–226, 2009.
doi: 10.1177/1094342009106189

[43] I. Cores, G. Rodríguez, M. J. Martín and P. Gonz'lez, "Reducing Application-level Checkpoint File Sizes: Towards Scalable Fault Tolerance Solutions," IEEE *10th International Symposium on Parallel and Distributed Processing with Applications*, 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," *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (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," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 501-514, 1 March 2019.
doi: 10.1109/TPDS.2018.2866794

[46] F. Li et al., "Checkpointing-Aware Loop Tiling for Energy Harvesting Powered Nonvolatile Processors," *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, pp. 15-28, Jan. 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," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1276-1288, 1 June 2022.
doi: 10.1109/TPDS.2021.3107745

[48] P. Sigdel, X. Yuan and N. -F. Tzeng, "Realizing Best Checkpointing Control in Computing Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 2, pp. 315-329, 1 Feb. 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," in *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337-350, Oct.-Dec. 2010.

[30] Sudarsun Kannan, Ada Gavrilovska, Karsten Schwan, and Dejan Milojicic. "Optimizing Checkpoints Using NVM as Virtual Memory." *In 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS'13),* pages 29–40, 2013.
doi: 10.1109/IPDPS.2013.69

[31] Lei Liu, Zehan Cui, Mingjie Xing, Yungang Bao, Mingyu Chen, and Chengyong Wu. "A Software Memory Partition Approach for Eliminating Bank-level Interference in Multicore Systems". *In Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*, pages 367–376, 2012.

[32] X. Liao, Z. Zhang, H. Liu and H. Jin, "Improving Bank-Level Parallelism for In-Memory Checkpointing in Hybrid Memory Systems," *in IEEE Transactions on Big Data*, vol. 8, no. 2, pp. 289-301, 1 April 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," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 12, pp. 2841-2853, Dec. 2013.
doi: 10.1109/TKDE.2013.35

[34] Btrfs, "Main Page," http://btrfs.wiki.kernel.org, 2013.

[35] C. -Y. Lin, L. -C. Wang and S. -P. Chang, "Incremental Checkpointing for Fault-Tolerant Stream Processing Systems: A Data Structure Approach," in *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, pp. 124-136, 1 Jan.-March 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*., 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," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1421-1433, 1 June 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," in *IEEE Computer*

doi: 10.1109/TDSC.2009.4

[50] M. Chtepen, F. H. A. Claeys, B. Dhoedt, F. De Turck, P. Demeester and P. A. Vanrolleghem, "Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids," *in IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 2, pp. 180-190, Feb. 2009. doi: 10.1109/TPDS.2008.93

[51] E. Deelman, et al., "Pegasus, a workflow management system for science automation," *Future Generation 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," in *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1105-1120, 1 Aug. 2018. doi: 10.1109/CLUSTER.2017.14

[54] A. Abdi, S. A. Asghari, H. Pedram, H. Taheri. "An Instruction Level Software Redundancy Based Method to Intra-Inter Block Control Flow Checking", *Soft Computing Journal*, 1(1), pp. 56-69, 2021. doi: 20.1001.1.23223707.1391.1.1.114.1

[55] E. Asadollahi, S. A. Asghari. "Prediction of Approperiate Number of Virtual Machines based on Time Series and Artifical Methoads via Virtual machines Clustering", *Soft Computing Journal*, 6(1), pp. 66-77, 2021.

[56] M. J. Nadjafi-Arani, S, Doostali. "Cost-based workflow scheduling using algebraic structures", Soft Computing Journal, 9(2), pp. 114-129, 2021. doi: 10.22052/scj.2021.242814.0

[57] M. Salehi, M. Khavari Tavana, S. Rehman, M. Shafique, A. Ejlali and J. Henkel, "Two-State Checkpointing for Energy-Efficient Fault Tolerance in Hard Real-Time Systems," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 7, pp. 2426-2437, July 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, Feb. 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. (ICESS)*, May 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," in *IEEE Transactions on Computers*, vol. 60, no. 5, pp. 639-652, May 2011, doi: 10.1109/TC.2010.129.