نشریه علمی- ترویجی محاسبات نرم
سال چهارم/ شمارهٔ هشتم/ پاییز و زمستان ۹۴/ صفحهٔ ۳۲ـ۵۳

دریافت مقاله: ۹۳/۱۲/۲۳
پذیرش مقاله: ۹۴/۸/۸

ارائهٔ یک رویکرد همانندسازی‌شدهٔ عامل‌محور در اجرای یک الگوی کد متحرک مطمئن

حجت‌الله حمیدی
گروه مهندسی فناوری اطلاعات، دانشگاه صنعتی خواجه نصیرالدین طوسی، تهران
H_hamidi@kntu.ac.ir

چکیده

عامل‌های سیار، امکان نزدیک کردن کدها به منابع را کـه در الگـوی قـدیمی مشـتری/خدمتگزار پـیش‌بینـی نشـده اسـت، امکان‌پذیر می‌کنند. انعطاف‌پذیری بیشتر الگوهای عامل سیار، در مقایسه با الگوی محاسبات مشـتری/خدمتگزار، هزینـه‌هـای اضافی به بار می‌آورد. پیچیدگی بیشتر، مسائلی همچون قابلیت اطمینان را مطرح می‌کند. در بین مکانیزم‌های قابلیت اطمینان، تحمل‌پذیری خطا، رویکردی است که از اهمیت قابل‌توجهی برخوردار است. رویکردهای مختلفی برای امنیـت عامـل‌هـای سیار در مقابل خطا و حمایت تراکنشی آن‌ها ارائه شده است. آن‌ها نقاط قوت و ضـعف مختلفـی داشـته و در محـیط‌هـای متفاوتی عمل می‌کنند و به‌علت این تنوع، اغلب برای برنامه‌نویس مشکل است که انتخاب کند کدام رویکرد، مناسـب‌تـرین رویکرد برای یک کاربرد است. این مقاله، یک تقسیم‌بندی از رویکردهای موجود برای اجرای عامل‌های سـیار تحمـل‌پـذیر خطا ارائه داده و درک ویژگی‌ها و ضعف‌های این رویکردها را آسان می‌کند. علاوه براین در این مقالـه، یـک سـاختار غیـر انسدادی برای امنیت عامل‌های سیار عامل‌محور ارائه شده است. در این ساختار، پروتکلی که تحمل‌پـذیری خطـا را فـراهم می‌سازد، با عامل حرکت می‌کند. مزیت این کار در این است که اجازه می‌دهد تا بتوان عامل سیار تحمل‌پذیر خطا را بـدون نیاز به تغییر دادن در طرح اصلی عامل اجرا کرد. مزایای این رویکرد، در مقایسه با رویکرد وابسته به محل، بسیار بیشتر بوده و نتایج حاصل، هزینه‌های تحمل‌پذیری خطا را در مقایسه با یک عامل همانندسازی‌نشده نشان می‌دهند. این هزینه‌ها به تعداد مراحل و اندازهٔ عامل بستگی دارد که افزایش اندازهٔ عامل و تعداد مراحل، زمان اجرا را افزایش می‌دهند.

واژه‌های کلیدی: عامل سیار(متحرک)، همانندسازی، عامل محور، تحمل‌پذیر خطا[1]، قابلیت اطمینان.

1. Fault-Tolerant

۱. مقدمه

یک عامل سیار، یک برنامهٔ کامپیوتری است که به‌صورت مستقل به‌جای کاربر عمل کرده و در طول یک شبکه از ماشین‌های ناهمگون حرکت می‌کند. با عامل‌های سیار، نزدیک کردن کد به منابع، که الگوی قدیمی مشتری/خدمتگزار آن را پیش‌بینی نکرده بود، امکان‌پذیر می‌شود؛ بنابراین عامل‌های سیار برای کاربردهای مختلفی از قبیل مدیریت شبکه و سیستم‌ها [۱ـ۵]، محاسبات سیار [۶]، بازیابی اطلاعات [۷]، تجارت الکترونیک [۸] در نظر گرفته شده‌اند. به‌علاوه، فناوری عامل سیار توجه تازه‌ای را در زمینهٔ محیط‌هایی که منابع محاسبات توزیع‌شده را برای کاربران فراهم می‌کنند، به خود جلب کرده است. در چنین محیط‌هایی، منابع روی ماشین‌های مختلف قابل دسترسی بوده، و کاربرد روی هر ماشینی که قادر به فراهم ساختن منابع مورد نیاز است، اجرا می‌شود. باوجوداین، این کار ممکن است انتقال کاربرد، یعنی حالت و کد آن به منابع را که در آنجا می‌تواند اجرا شود، بطلبد. این دقیقاً همان مشکلی است که فناوری عامل سیار سعی می‌کند آن را حل کند. انعطاف‌پذیری بیشتر الگوی عامل سیار، در مقایسه با الگوی محاسبات مشتری/خدمتگزار، هزینه‌های اضافی به بار می‌آورد. این هزینه‌ها در کنار هزینه‌های دیگر، پیچیدگی بیشتر و مشکلات ایجاد و مدیریت کاربردها در این زمینه را شامل می‌شود [۹ـ۱۱]. پیچیدگی بیشتر، مسائلی همچون قابلیت اطمینان را مطرح می‌کند. قبل از آنکه فناوری عامل سیار بتواند در مرکز کاربردهای تجاری قرار گیرد، باید مکانیزم‌های قابلیت اطمینان برای عامل‌های سیار ایجاد شوند. در بین این مکانیزم‌های قابلیت اطمینان، تحمل‌پذیری خطا و حمایت تعاملی، مکانیزم‌هایی هستند که اهمیت قابل‌توجهی دارند. خرابی‌ها در یک سیستم عامل سیار، ممکن است به ناپدید شدن یا از بین رفتن جزئی یا کامل عامل بینجامد. برای به‌دست‌آوردن تحمل‌پذیری خطا، مالک عامل یعنی شخص یا ایجادکنندهٔ کاربرد و شکل‌دهندهٔ عامل، می‌تواند برای ردیابی خطای عامل خود تلاش کرده و به هنگام چنین اتفاقی، یک عامل جدید را راه‌اندازی کند. باوجوداین، این کار آشکارسازی صحیحِ خرابی عامل، یعنی تشخیص یک عامل ازکارافتاده از عاملی را که به‌علت پردازشگرهای کند با پیوندهای ارتباطی کند تأخیر کرده است، می‌طلبد. متأسفانه این کار در سیستم‌هایی مانند اینترنت نمی‌تواند عملی شود تا در جایی که به اشتباه فکر می‌کند که عامل از کار افتاده است، از این حالت جلوگیری کند. در این حالت به کار انداختن یک عامل جدید به اجرای چندبارهٔ عامل، یعنی نقض ویژگی اجرای فقط یک بار عامل منجر خواهد شد. اگرچه انجام این کار (راه‌اندازی یک عامل جدید) ممکن است در برخی کاربردها مورد قبول واقع شود (برای مثال، کاربردهایی که عملیات آن‌ها عوارض جانبی نداشته باشد یعنی خود توان)، کاربردهای دیگر را از این کار منع می‌کنند. خوشبختانه در محیط‌هایی با ردیابی نامطمئن خرابی، می‌توان با تکرار عامل‌ها (همانندسازی عامل‌ها) با یک پروتکل مناسب از اجرای چندبارهٔ عامل جلوگیری کرد. واقع امر این است که اگر ردیابی خرابی نامطمئن باشد، تکرار (همانندسازی) به‌تنهایی نمی‌تواند ویژگی اجرای فقط یک بار را تضمین کند. در بخش دوم، مزایای الگوی عامل سیار و دامنه‌های کاربرد عامل‌های سیار ارائه می‌شوند. در بخش سوم، اجرای عامل سیار تحمل‌پذیر خطا، مدل عامل سیار، مدل خرابی‌ها و مدل همانندسازی ارائه شده‌اند. یک خرابی زمانی اتفاق می‌افتد که یک جزء سخت افزاری یا نرم افزاری در سیستم از کار بیفتد. برعکس، خرابی معنایی در صورتی اتفاق می‌افتد که یک سرویس فراهم نشود؛ اگرچه ممکن است هیچ از کار افتادگی روی نداده باشد. برای مثال، درخواست یک رزرو صندلی در یک هواپیمای پر به خرابی معنایی می‌انجامد. در نهایت، عامل‌های سیار تحمل‌پذیر خطای تعاملی و غیرتعاملی از یکدیگر متمایز می‌شوند. در بخش چهارم، دسته‌بندی رویکردهای عامل سیار تحمل‌پذیر خطا مشخص می‌شوند. در بخش پنجم، رویکردهای اجرای عامل سیار تحمل‌پذیر خطا ارائه می‌شوند. در بخش ششم، انتشار مطمئن و مسائل الگوریتمی ارائه شده‌اند. در بخش هفتم، رویکرد وابسته به عامل و در بخش هشتم، ارزیابی

عملکرد و شبیه‌سازی‌ها ارائه شده‌اند و در نهایت در بخش نهم، نتیجه‌گیری مقاله ارائه شده است.

۲. مزایای الگوی عامل سیار

Chess و همکارانش [۲] تلاش کرده‌اند تا فواید تکنولوژی عامل سیار را برآورد کنند. آن‌ها به این نتیجه رسیدند که تمام مسائلی که به‌عنوان نمونه‌های خوب برای استفاده از عامل‌های سیار در نظر گرفته می‌شوند، می‌توانند با استفاده از راه‌حل‌های سنتی مشتری/خدمتگزار حل کنند. بااین‌حال، عامل‌های سیار یک راه‌حل عمومی برای چنین مسائلی ارائه می‌کنند؛ یعنی به‌جای ایجاد راه‌حل‌های مختلف و منطبق با هر مسئله، عامل‌های سیار یک راه‌حل عمومی برای این مسائل فراهم می‌سازند، الگوی محاسبات عامل سیار دارای چندین مزیت نسبت به مدل قدیمی مشتری/خدمتگزار است [۱۱ـ۱۹] که مهم‌ترین آن‌ها عبارت‌اند از:

تأخیر ارتباطات و پهنای باند: Sabel و همکارانش مدلی را برای اندازه‌گیری‌های عملکرد در [۲۰] معرفی می‌کنند. آن‌ها اظهار می‌کنند که در الگوی مشتری/خدمتگزار، هزینهٔ کلی بر همکنش به اندازهٔ (بزرگی) بانک اطلاعاتی بستگی دارد. از سوی دیگر، هزینهٔ کلی الگوی عامل سیار، وابسته به اندازهٔ کد و اطلاعات فرستاده‌شده توسط کاربرد است. این مسئله به این نتیجه منجر می‌شود که به‌کار بردن الگوی عامل سیار تنها زمانی توجیه‌پذیر است که اندازهٔ بانک اطلاعاتی بیشتر از حد آستانه باشد. برای بهره بردن از محلی بودن، ضروری است که ایجادکنندهٔ کاربردهای عامل سیار، ایدهٔ محلی بودن را با هر سرور همراه کند.

حفاظت از پروتکل‌ها: وقتی داده‌ها در یک سیستم توزیع‌شده تبادل می‌شوند، هر میزبان صاحب کدی است که پروتکل‌هایی را پیاده‌سازی می‌کند که برای کد کردن مناسب داده‌های خروجی و ترجمهٔ داده‌ها ورودی لازم هستند. عامل‌های سیار می‌توانند به میزبان‌های دور منتقل شوند تا کانال‌هایی را ایجاد کنند که بر مبنای پروتکل‌ها عمل کنند.

تحمل‌پذیری خطا: توانایی عامل‌های سیار، برای واکنش نشان دادن به‌صورت پویا به شرایط و رویدادهای نامناسب، امکان ایجاد سیستم‌های تحمل‌پذیر در برابر خطا را میسر می‌سازد. اگر یک میزبان از دور خارج شود، به تمامی عامل‌هایی که روی آن در حال اجرا هستند، هشدار داده شده و به آن‌ها فرصت زمانی برای انتقال به یک میزبان دیگر بر روی شبکه و ادامهٔ عملیات داده خواهد شد.

مدیریت خطا: مدیریت خطا عبارت‌اند از تشخیص خطا در ابزارهای متصل به شبکه. در این خصوص نیز عامل‌ها می‌توانند به‌نحو مطلوبی مورد استفاده قرار گیرند. عامل می‌تواند با مشاهدهٔ میزان استفاده از یک منبع محلی و کنترل حجم استفاده از آن، در کنترل خطا بسیار مؤثر باشد.

۱.۲. دامنه‌های کاربرد عامل‌های سیار

تجارت الکترونیکی: استفاده از تکنولوژی عامل سیار برای فراهم آوردن برخی خدمات تجارت الکترونیکی پیشنهاد شده است. به‌طور خاص‌تر، عامل‌ها به اینترنت وارد می‌شوند و اطلاعات مورد نیاز را جمع‌آوری می‌کنند. در صورت نیاز آن‌ها می‌توانند با عامل‌های دیگر که به اشتراک، مبادله یا خریدن اطلاعات مشغول‌اند، همکاری کنند [۲۱]. طرح‌هایی ارائه می‌شوند که بازارهایی را ایجاد می‌کند که عامل‌ها می‌توانند در آن‌ها معامله کنند و حتی در مزایده‌ها شرکت کنند. برای رسیدن به این هدف، عامل با مکانیزم‌هایی برای تحویل پول به کالاها یا خدمات خریداری‌شده به‌جای متولی، مجهز می‌شود [۲۲ـ۲۳].

عملیات‌های سیار: تعامل‌ها ممکن است شامل چند گره باشند. اگر قرار باشد که گره‌ها به‌ترتیب، و نه به‌صورت موازی، ملاقات شوند، یک عامل می‌تواند یک جایگزین برای اجرای یک دستور از نوع مشتری/خدمتگزار برای هر میزبان باشد. این زمینهٔ کاربردی به تجارت الکترونیکی، مثلاً خرید مقایسه‌ای مربوط می‌شود [۲۴ـ۲۶].

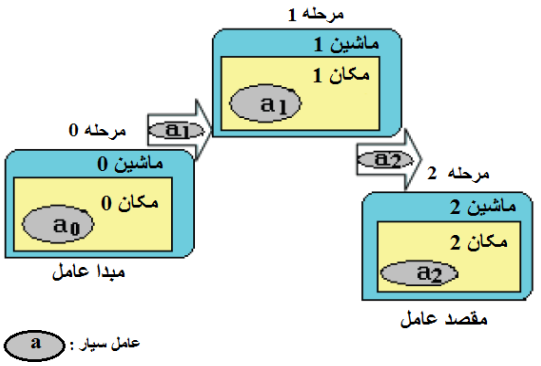
جمع‌آوری اطلاعات: درصورتی‌که مجبور به در نظر گرفتن چند مبدأ یا مبدأهای از قبل تعیین شده به‌صورت دقیق باشیم، مسئولیت جمع‌آوری اطلاعات به عامل سپرده می‌شود. این عامل در طول مسیر حرکت خود، مقداری اطلاعات جمع‌آوری می‌کند. این اطلاعات اجازهٔ تصمیم‌گیری در مورد مسیر حرکت بعدی را به او می‌دهد [۲۴].

پـردازش مـوازی: عامـل‌هـای سـیار مـی‌تواننـد از خـود، همانندسازی کرده و کار را بین نسخه‌های خود تقسیم کنند. ایـن کار به عامل‌های سیار اجازه مـی‌دهـد تـا در پـردازش مـوازی و توزیع‌شده بین گره‌های مختلف اجرا شوند؛ بنابراین، یک کـاربرد مبتنی بر تکنولوژی عامل سیار، بسیار آسان‌تر از کاربردهـایی کـه از بلوک‌های یکپارچه تشکیل مـی‌شـوند و قابـل انـدازه‌گیـری و ارزیابی‌اند. بااین‌حال، پـیش از آنکـه عامـل‌هـای سـیار در یـک مقیاس بزرگ ظـاهر شـوند، طـرح‌هـای عامـل‌هـای سـیار بایـد سرویس‌های زیرسـاختی را بـرای تسـهیل ایجـاد عامـل فـراهم سازند. در بین این سرویس‌ها امنیت، ادارهٔ عامل‌ها، تحمل‌پذیری خطا و حمایت تعاملی قرار دارند. تحمل‌پذیری خطا و حمایـت تعاملی موارد تأکید این مقاله‌اند. به‌علاوه، ابعاد تکنولـوژی عامـل باید به‌صورت استاندارد درآیند تا به سیستم‌های عامـل مختلـف اجازه دهد تا بین خودشان کار کنند (فعالیت بین سیستمی داشـته باشند). اگرچه تلاش‌های استانداردسازی در حـال ایجـاد شـدن هستند یا قبلاً وجود داشته‌اند، هنوز به‌صورت گسترده به خدمت گرفته نشده‌اند.

۳. اجرای عامل سیار تحمل‌پذیر خطا

۱.۳. مدل عامل سیار

یک عامل سیار روی یکسری از ماشین‌ها اجرا می‌شـود کـه در آن، محل P_i محیط منطقی اجرا را برای عامـل فـراهم مـی‌کنـد (شکل ۱). اجرای عامل در یک محل یک مرحله، S_i از اجـرای عامل نامیده می‌شود. محل‌هایی را که اولـین و آخـرین مرحلـهٔ عامل در آن‌ها اجرا می‌شود، مبدأ و مقصد عامل می‌نامند.



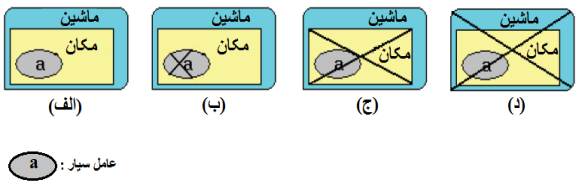
شکل (۱): مدل اجرای عامل سیار با دو مرحله [۱]

به‌طور منطقی، یک عامل سیار در طی یـک سـری اعمـال، اجرا می‌شود. هـر عمـل بـا یـک عامـــل a_i $(0 \le i \le n)$ در مرحله‌های معادل S_i نمایش داده می‌شود. محل p_i، عامل a_i را در مرحلهٔ s_i اجرا می‌کند که به یک حالت درونی جدید عامل، همچنین به یک حالت جدید بالقوه از محل می‌انجامـد. عامـل حاصل‌شده، a_{i+1} نامیده شده، محل p_i، a_{i+1} را بـه P_{i+1} (کـه $i<n$) به جلو هدایت می‌کند.

۲.۳. مدل خرابی‌ها

۱.۲.۳. خرابی‌های زیرساختی

هریک از اجزای سخت‌افزاری یـا نـرم‌افـزاری در یـک محـیط محاسبه، به‌صورت بالقوه در معرض خرابی[1] قـرار دارد (شـکل ۲ـالف، عامل سیار در حالت بـدون خطـا را نشـان مـی‌دهـد.) ماشین‌ها، محل‌ها و عامل‌هـا مـی‌تواننـد خـراب شـوند. کـاملاً واضح است که از کار افتادگی[2] یک ماشین باعث می‌شـود هـر محل و عاملی که روی این ماشین فعالیت می‌کنـد نیـز از کـار می‌افتد (شکل ۲ـ د). یک محل از کار افتاده نیـز باعـث از کـار افتادن هر عاملی روی این محل مـی‌شـود، امـا بـه‌طـور کلـی، تأثیری روی ماشین نمی‌گذارد (شکل ۲ـ ج). به همین ترتیـب، در صورت از کار افتادن عامل، ماشین و محل از کار نمی‌افتنـد (شکل ۲ـ ب)؛ به عبادت دیگر از کار افتادگی عامل تأثیری بـر روی ماشین و محل نمی‌گذارد.



شکل (۲): مدل خرابی برای عامل‌های سیار

بنـابراین خرابـی یـک محـل باعـث خـراب شـدن تمـام عامل‌هایی می‌شود که روی آن محل اجرا می‌شوند و به همـین ترتیب، خرابـی یـک ماشـین باعـث خرابـی تمـام محـل‌هـا و عامل‌های روی آن ماشین می‌شوند. همان‌طور کـه اشـاره شـد، اجرای یک عامل روی یک محـل عمومـاً بـه تغییـر در حالـت

1. Failure
2. Crash

عامل و حالت محل می‌انجامد. به‌علت وابستگی خاصی که بین خرابی عامل و محل وجود دارد، خرابی عامل ممکن است باعث شود که ماشین و محل به یک حالت نادرست درآیند؛ برای مثال فرض کنید که عامل، پول را از یک خدمات بانکی فراهم‌شده توسط محل، به خود آن محل انتقال می‌دهد؛ اگر عامل در این نقطه از کار بیفتد، حالت محل عملیات عامل را منعکس می‌کند (بازیابی پول)، که در آن حالت عامل و خود پول ناپدید شده‌اند. واضح است که این عمل به حالت نادرست محل می‌انجامد. درنتیجه، عملیات‌های عامل سیار روی یک محل باید قابل خنثی کردن باشند؛ پس باید راه‌های مناسبی تهیه شوند تا حالت پایدار ماشین و محل راحتی زمانی که عامل از کار می‌افتد، تضمین کند [۲۷ـ۲۸].

۲.۲.۳. خرابی‌های معنایی

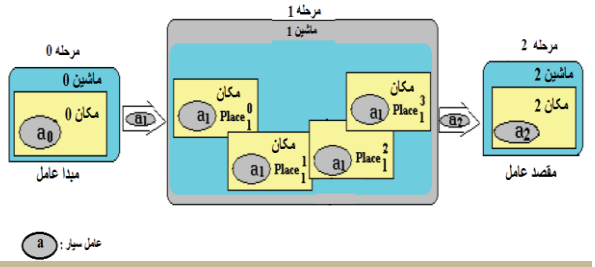
یک خرابی معنایی از این نظر با خرابی‌های زیرساختی متفاوت است که نه ماشین، محل یا عامل که اجرای درخواست را شروع می‌کنند، از کار نمی‌افتند، بلکه آن زمانی اتفاق می‌افتد که سرویس مورد نیاز به‌دلیل خرابی سرویس، تحویل داده نمی‌شود؛ برای مثال، یک درخواست برای یک بلیط هواپیما ارائه می‌شود، درصورتی‌که هیچ صندلی خالی در این پرواز موجود نیست. بااین‌حال، عملیات عامل، یعنی درخواست بلیط، به‌طور کامل اجرا می‌شود. در حقیقت، در این مثال، هیچ خرابی واقعی رخ نداده است. بااین‌حال از دیدگاه عامل (مشتری آن سرویس)، نتیجهٔ درخواست سرویس، نامطلوب است؛ این نتیجه یک نتیجهٔ خرابی معنایی نامیده می‌شود [۲۹].

۳.۳. مدل همانندسازی

تحمل‌پذیر خطا می‌تواند از طریق شکلی از همانندسازی به‌دست آید. هدف تحمل‌پذیری خطا (۱)تغییر دادن یک رفتار غیرقابل پیش‌بینی یا نامطلوب به یک رفتار قابل پیش‌بینی یا (۲) پوشاندن رفتار غیرقابل پیش‌بینی یا نامطلوب است. مورد (۱) معمولاً با استفاده از تعامل‌های اتمیک به‌دست می‌آید [۳۰]، درحالی‌که حالت (۲) کلاً از همانندسازی استفاده می‌کند. تحمل‌پذیری خطا فقط از طریق معرفی افزونگی[1] به سیستم به‌دست می‌آید، درحالی‌که استفاده از افزونگی در حالت (۲) کاملاً آشکار است، اما در حالت (۱) تا حدی پنهان است؛ برای مثال، استفاده از تعامل‌ها به اطلاعات افزونه تکیه دارد. شکل‌های مختلفی از افزونگی وجود دارند که برخی از آن‌ها عبارت‌اند از: افزونگی سخت‌افزاری، افزونگی نرم‌افزاری، افزونگی زمانی، افزونگی اجرایی، افزونگی داده‌ای (اطلاعاتی). با افزونگی اطلاعاتی، نسخه‌هایی از داده‌ها در موقعیت‌های مختلف ذخیره می‌شوند. درصورتی‌که یک نسخهٔ خراب ناپدید شود، می‌توان به یک نسخهٔ دیگر دسترسی پیدا کرد. افزونگی اجرا، برعکس، اجرای یک برنامه را نسخه‌برداری می‌کند. در صورت خرابی یک نسخه از برنامه، نسخهٔ دیگر اجرا می‌شود. در محاسبات توزیع‌شده، اغلب، سرور برای به‌دست آوردن تحمل‌پذیری خطا نسخه‌برداری می‌شود.

همانندسازی از انسداد جلوگیری می‌کند. به جای فرستادن عامل به یک محل در مرحلهٔ بعدی، نسخه‌های عامل به یک مجموعهٔ M_i، از محل‌های $p_i^0, p_i^1, p_i^2, ...$ فرستاده می‌شوند (شکل ۳). نسخه‌ای از عامل a_i که بر روی محل p_i^j قرار دارد، a_i^j نامیده می‌شود، و به‌منظور جلوگیری از انسداد، عامل در مرحلهٔ s_i به مجموعهٔ محل‌های M_{i+1} در مرحلهٔ S_{i+1} فرستاده می‌شود؛ به عبارت دیگر، محل p_i^j میزبان نسخهٔ a_i^j از عامل a_i می‌شود. درصورتی‌که یک محل $p_{i+1}^k \in M_{i+1}$ به هنگام اجرای عامل از کار بیفتد، یک محل دیگر $p_{i+1}^j \in M_{i+1} (j \neq k)$ این کار را برعهده می‌گیرد. شکل (۳) یک مثال از اجرای عامل سیار را نشان می‌دهد. برای جلوگیری از انسداد، عامل در مرحلهٔ s_i به مجموعه محل‌های M_{i+1} در مرحلهٔ S_{i+1} فرستاده می‌شود به‌جای اینکه یک محل P_{i+1} فرستاده شود؛ به عبارت دیگر، محل P^j_i میزبان نسخهٔ a^j_i از عامل a_i می‌شود. اگر یک محل $P^k_{i+1} \in M_{i+1}$ به هنگام اجرای عامل از کار بیفتد، یک محل دیگر $P^k_{i+1} \in j=k$ M_{i+1} این کار را بر عهده می‌گیرد. شکل (۳) یک مثال از اجرای عامل سیار را نشان می‌دهد.

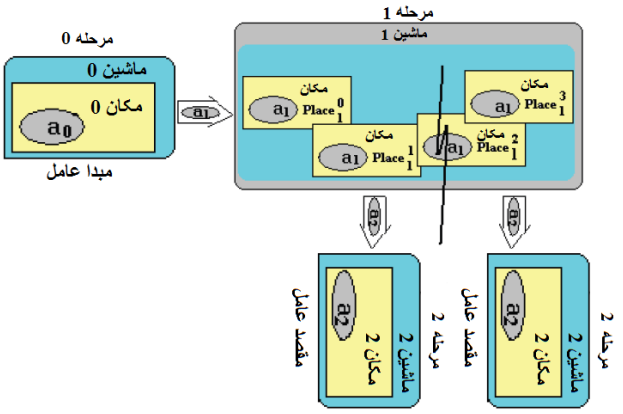
1. Redundancy



شکل (۳): یک مثال از اجرای عامل سیار با دو محل اضافه‌شده [۱]

افزونگی در اجرای مرحله‌ای، خرابی‌های سیستم را می‌پوشاند و عامل را قادر می‌سازد تا علی‌رغم وجود خرابی‌ها، همان‌طور که در شکل (۳) مشاهده می‌شود، به‌جای فرستادن عامل به یک محل p_1^0 در مرحلهٔ بعد، عامل به مجموعهٔ M_i از محل‌های $\{p_i^0, p_i^1, p_i^2, ...\}$ فرستاده می‌شود. افزونگی در مبدأ و مقصد عامل، لازم نیست زیرا در مبدأ، عامل شروع به کار می‌کند و حالت خود را می‌خواند؛ بنابراین هنوز تحت کنترل مالک عامل قرار دارد. و مقصد عامل محلی است که در آن عامل نتایج خود را در دسترس مالک قرار می‌دهد. اگر یک محل از کار بیفتد (مانند P^2_1 در شکل ۴)، اجرای عامل مسدود نمی‌شود و محل‌های P^3_1 , P^0_1, P^1_1 می‌توانند اجرای عامل a_1 را به‌عهده گرفته و از انسداد جلوگیری کنند. در واقع مالک عامل این تضمین را دارد که عامل در نهایت، اجرای خود را به پایان خواهد رساند. قابل ذکر است که در مراحل S_0 و S_2، همانندسازی صورت نمی‌گیرد، زیرا عامل تنها در مبدأ عامل شکل می‌گیرد و نتایج را در مقصد عامل به مالک عامل ارائه می‌کند؛ بنابراین، در این مراحل (S_0وS_2) همانندسازی مورد نیاز نیست. علی‌رغم همانندسازی عامل، تقسیم‌بندی‌های شبکه می‌توانند هنوز هم از پیشروی عامل جلوگیری کنند. در واقع، اگر شبکه به شکلی تقسیم شود که تمام محل‌های در حال اجرای عامل در مرحلهٔ S_i، روی یک جزء و محل‌های مرحلهٔ S_{i+1} در یک جزء دیگر قرار گیرند، عامل نمی‌تواند اجرای خود را ادامه دهد.

در شکل (۴)، تمام محل P^2_1 در مرحلهٔ S_1 خراب شده است. باوجوداین، اجرای عامل می‌تواند در محل‌های P^3_1، P^0_1, P^1_1 ادامه یابد، با اینکه محل P^2_1 از کار افتاده است.



شکل (۴): اجرای عامل با محل‌های افزوده‌شده که در آن، محل P^2_1 خراب شده است، محل‌های افزوده‌شده (یعنی P^3_1 ، P^0_1, P^1_1) خرابی آن محل را می‌پوشانند.

۴. دسته‌بندی رویکردهای عامل سیار تحمل‌پذیر خطا

تعامل محلی به‌عنوان جزء ساختار اصلی اجرای عامل سیار تحمل‌پذیر خطا (برای حل خرابی های زیرساخت) است. عملیات‌های مرحله‌ای عامل سیار به‌صورت تعامل‌های محلی، اجرا می‌شوند. زمانی که عملیات‌های مرحله‌ای اجرا شوند، تعامل‌های محلی نیز یا اجرا می‌شوند یا متوقف می‌شوند. این تصمیم مبنی بر اینکه کدام تعامل محلی اجرا می‌شود و کدام تعامل محلی ناتمام بماند، تصمیم اجرا[1] (یا رویکرد CD) نامیده می‌شود؛ به عبارت دیگر، تصمیم اجرا، اجرای فقط یک بار را توسط اجرای فقط یک بار تعامل محلی در یک مرحله، و متوقف کردن تمام تعامل‌های دیگر تضمین می‌کند. این فرایند می‌تواند در زمان‌های مختلفی از اجرای عامل سیار رخ دهد: ۱. در پایان اجرای مرحله‌ای (که اجرای پس از مرحله[2] یا رویکرد CAS نامیده می‌شود.) ۲. در پایان اجرای عامل سیار، یعنی در مقصد عامل (که اجرای در مقصد[3] یا رویکرد CAD نامیده می‌شود.) درصورتی‌که در حالت (۲) فقط یک بار در تمام طول اجرای عامل سیار تصمیم گرفته می‌شود، در حالت (۱) برای هر مرحله یک تصمیم مورد نیاز است. اگرچه در حالت (۲) تصمیم اجرا، ممکن است فقط در مقصد عامل رخ

1. Commit decision: CD
2. Commit – after – Stage: CAS
3. Commit – at – destination :CAD

دهد، هدف آن باید ویژگی فقط یک بار را در مراحلی که بیش از یک نسخهٔ عامل اجرا شده، تضمین کند.

۱.۴. رویکردهای اجرای پس از مرحلهٔ CAS

رویکردهای مرحلهٔ CAS، عملیات‌های هر مرحله را در پایان هر مرحلهٔ S_i، قبل از آنکه عامل به مرحلهٔ بعدی S_{i+1} منتقل شود، اجرا می‌کنند. درصورتی‌که اجرای عامل سیار در مرحلهٔ S_i نسخه‌برداری شود، این رویکرد اهمیت خاصی پیدا می‌کند (شکل ۴). به‌طور خاص‌تر، تصمیم اجرا (CD) این امکان را می‌دهد تا از اجرای مکرر عامل جلوگیری شود؛ بنابر این ویژگی «فقط یک بار» را تضمین می‌کند. در واقع اجرا فقط یک بار روی یک محل از یک مرحله و توقف اجراهای دیگر (در صورت وجود)، اجرای فقط یک بار عامل را تضمین می‌کند. به این منظور لازم است که این دو حالت از یکدیگر متمایز شوند:

۱. اجرای عامل a_i روی یک محل واحد (اجرای عامل نسخه‌برداری‌نشده).

۲. اجرای عامل a_i روی مجموعه‌ای از محل‌های M_i(اجرای عامل نسخه‌برداری‌شده).

علاوه بر این، تصمیم اجرا می‌تواند توسط یک محل یا توسط چند محل اتخاذ شود. در نهایت، تصمیم اجرا می‌تواند با اجرای عامل a_i ترکیب شود یا نشود. ترکیب ۳ معیار زیر به ۸ راه‌حل منتهی می‌شود که در ادامه به بررسی آن‌ها پرداخته می‌شوند:

۱. موقعیت اجرای عامل

۲. موقعیت تصمیم اجرا

۳. متمرکزشده- توزیع‌شده

SSC[1]: راه‌حل SSC رویکردهایی را که یک عامل روی محل واحد P_i اجرا کرده و سپس عامل به محل بعدی P_{i+1} انتقال یافته، شامل می‌شود. در حقیقت، نتیجهٔ تصمیم اجرا، همیشه اجراست. هرگز تصمیم توقف اتخاذ نمی‌شود، زیرا هیچ دلیلی برای توقف اجرای عامل از دیدگاه محل P_i وجود ندارد. علاوه بر این درصورتی‌که P_i بازیابی نشود، عامل (کد و حالت آن) از بین می‌روند. جالب اینکه ناپدید شدن و از بین رفتن عامل نیز به انسداد می‌انجامد، زیرا مالک عامل منتظر بازگشت عامل می‌ماند. رویکرد SSC اغلب در محیط‌هایی مورد مطالعه قرار می‌گیرد که در آن‌ها خرابی‌ها به‌ندرت اتفاق می‌افتند یا انسداد در آن‌ها مشکلی به‌وجود نمی‌آورد، به این علت که اجزای خراب‌شده به‌سرعت ترمیم می‌شوند. علاوه‌براین به شرط استفاده از یک مکانیزم بازیابی مناسب، ویژگی اجرای فقط یک بار تضمین می‌شود.

SMC[2]: همانند راه‌حل SSC، در این راه‌حل، عملیات مرحله‌ای عامل a_i فقط بر روی یک محل p_i اجرا می‌شود. برعکس، تصمیم اجرا، در چندین محل توزیع می‌شود. درنتیجه، یک خرابی در محل p_i به انسداد اجرای عامل سیار می‌انجامد. از سوی دیگر، از آنجایی که تصمیم اجرا توزیع می‌شود، پس غیر انسدادی است. توزیع تصمیم اجرا درحالی‌که اجرا فقط روی یک محل روی می‌دهد، عجیب به‌نظر می‌رسد؛ بنابراین راه‌حل SMC به‌ندرت مورد استفاده قرار می‌گیرد. راه‌حل SMC ممکن است برای پروتکل‌هایی که در آن‌ها عامل سیار باید روی محل‌های بسیار خاصی اجرا شود، قابل استفاده باشد؛ برای مثال، مالک عامل ممکن است بخواهد که فقط به خطوط هوایی ایرلاین پرواز کند که فقط یک سرویس غیر نسخه‌برداری‌شده یا غیر همانندسازی‌شده را حمایت می‌کند.

MSC[3]: در این رویکرد، عملیات مرحله‌ای عامل a_i توسط چندین محل اجرا شود، درحالی‌که تصمیم اجرا توسط یک محل p_i^k آغاز می‌شود. در راه‌حل MSC، تصمیم اجرا، محلی را که عامل را اجرا کرده است؛ یعنی محل اصلی p_i^{prim} را معین می‌کند. محل اصلی، محلی در یک مرحله است که تعامل محلی عامل a_i را بر عهده می‌گیرد، درحالی‌که محل‌های دیگر، یعنی محل‌های غیر از محل اصلی، تمام تغییرات عامل a_i را متوقف می‌کنند، این کار به ما اجازه می‌دهد تا از اجراهای چند بار عامل a_i جلوگیری کرده و از نقض ویژگی فقط یک بار جلوگیری کنیم.

باوجوداینکه اجرای عملیات مرحله‌ای، غیر انسدادی است،

1. Single- Single- Collocated: SSC

2. Single – Multiple – Collocated:SMC

3. Multiple – Single - Collocated

انسداد ممکن است که در پروتکل اجرا رخ دهد. علت آن این است که تنها یک محل، پروتکل را اجرا می‌کند. که اگر این محل خراب شود، تصمیم اجرا و درنتیجه، کل اجرای عامل سیار مسدود می‌شود.

MMC[1]: راه‌حل MMC تعمیمی از رویکردهایی است که در آن‌ها اجرای عملیات مرحله‌ای و تصمیم اجرا باهم تلفیق می‌شوند؛ به عبارت دیگر، اجرا و تصمیم اجرا، هر دو روی چندین محل توزیع می‌شوند. این توزیع از انسداد دوری می‌کند، اما به نقض ویژگی اجرای فقط یک بار منجر می‌شود. برای حفظ ویژگی فقط یک بار، لازم است محل‌هایی که عامل را اجرا کرده‌اند، موافقت وجود داشته باشد. این محل‌ها عبارت‌اند از: ۱. محل اصلی p_i^{prim} که تغییرات انجام‌شده توسط عامل را بر عهده می‌گیرد؛ ۲. محل‌های دیگری که این تغییرات را متوقف می‌کنند. به عبارت دیگر، از اجرای چند بار عامل سیار، نمی‌توان جلوگیری کرد مگر اینکه یک موافقت به‌دست آید، یعنی اینکه یک مسئله موافقت حل شود.

SSC[2]: راه‌حل SSD مشابه SSC است، به‌جز اینکه در این راه‌حل اجرای عامل و تصمیم اجرا توزیع می‌شود؛ به عبارت دیگر محلی که عامل را اجرا می‌کند و محلی که تصمیم اجرا را اتخاذ می‌کند یکسان نیستند، بلکه هر محل p_k می‌تواند تصمیم اجرا اتخاذ کند و سپس باید به محل p_i منتقل شود. این انتقال در معرض خرابی‌هایی همانند ناپدید شدن یا از بین رفتن پیام قرار دارد. به‌علاوه تفکیک اجرای عامل a_i و تصمیم اجرا به‌شدت، تحمل‌پذیری خطای اجرای عامل را تضعیف می‌کند. این احتمال که P_i و P_K خراب نشوند، کمتر از این احتمال است که p_i خراب نشود. درنتیجه، احتمال انسداد بیشتر است و این راه‌حل به اندازهٔ SSC قابل‌توجه نیست.

SMD[3]: این راه‌حل همانند SMC است.

MSD[4]: یک مجموعه از محل‌ها، عملیات مرحله‌ای عامل a_i را اجرا می‌کنند، درحالی‌که تصمیم اجرا روی هر تک‌محل p_k واقع شده است. اجرای عملیات مرحله‌ای، غیر انسدادی است، اما تصمیم اجرا می‌تواند مسدود شود. خرابی‌ها در کانال ارتباطی بین p_k و محل‌هایی که عمل مرحله‌ای a_i را اجرا می‌کنند نیز ممکن است به انسداد منجر شوند.

MMD[5]: برای جلوگیری از مسئلهٔ انسداد، اجرای عملیات مرحله‌ای عامل a_i و تصمیم اجرا، هر دو به یک مجموعهٔ جدا از محل‌ها توزیع می‌شوند. تفاوت اصلی این راه‌حل با راه‌حل MMC آن است که MMC می‌تواند از محلی بودن تصمیم اجرا و اجرای عامل a_i استفاده کند. به‌علاوه، MMC دچار خرابی‌های ارتباطی بین محل‌های اجراکنندهٔ عمل مرحله‌ای (p_i^j) و محل‌های اجراکنندهٔ پروتکل اجرا نمی‌شوند.

۲.۴. رویکردهای CAD

برخلاف رویکردهای CAS، در این رویکردها عملیات‌های مرحله‌ای عامل سیار، فقط در پایان اجرای عامل، اجرا می‌شوند، درحالی‌که در رویکردهای CAS، عامل‌های همانندسازی‌شده[1] آشکار شده و در اجرای مرحله کنار گذاشته می‌شوند. در رویکردهای CAD، این عامل‌های همانندسازی‌شده به اجرای خود ادامه می‌دهند. نسخه‌برداری‌های افزوده‌شده، فقط در یک محل مشترک که در آن نسخه‌برداری‌ها و عامل اصلی به هم می‌رسند، آشکارسازی می‌شوند. در کل، محل مشترک فقط مقصد عامل است؛ بنابراین، مقصد عامل تنها محلی است که تصمیم صحیح دربارهٔ این را که چه عاملی اجرا شود و کدام یک کنار گذاشته شود، اتخاذ می‌کند. معمولاً اولین عامل که به مقصد می‌رسد، اجرا می‌شود، درحالی‌که عامل‌های افزوده‌شده که بعداً می‌رسد، کنار گذاشته می‌شوند. و عملیات‌های مرحله‌ای آن‌ها کنار گذاشته می‌شود. این کار به ما اجازه می‌دهد که ویژگی فقط یک بار را برای اجرای عامل سیار تحمل‌پذیر خطا تضمین کند تا زمانی که عامل به مقصد نرسیده باشد، تعامل‌های محلی اجرا نخواهند شد یا به عبارت دیگر متوقف می‌شوند؛ البته باقی می‌مانند تا نتیجه اجرای عامل تعیین شود.

1. Multiple – Multiple - Collocated
2. Single – Single - Collocated
3. Single – Multiple – distributed
4. Multiple – Single – distributed
5. Multiple – Multiple – distributed

SSD[1]: این راه‌حل مشابه راه‌حل SSD برای رویکردهای اجرا پس از مرحلهٔ CAS است با این تفاوت که اجرا، برای تمام عملیات‌های مرحله‌ای، در مقصد عامل رخ می‌دهد. این راه‌حل، یک راه‌حل انسدادی بوده و از سوی دیگر از اجرای عامل‌های نسخه‌برداری‌شده نیز جلوگیری می‌کند. واضح است که رویکرد اجرای در مقصد SSD، فقط از لحاظ تئوریک جالب است. ازآنجایی‌که تصمیم اجرا، همیشه اجرا می‌شود، تعامل‌های محلی می‌توانند به‌سرعت پس از اجرای مرحله، اجرا شوند (مشابه حالت اجرای پس از مرحلهٔ SSD)، به‌جای اینکه منتظر بمانند تا عامل به مقصدش برسد. علت پیشنهاد این راه‌حل این است که کمک می‌کند تا تفاوت بین اجرای عامل سیار تعاملی و غیر تعاملی بهتر درک شود.

MSD[2]: برخلاف رویکرد SSD، در این راه‌حل، با اجرای عامل روی محل‌های چندگانه، در صورت نیاز از انسداد جلوگیری می‌شود. ازآنجایی‌که محل‌های قبلی، یک نسخه از عامل را از قبل دارند، معمولاً زمانی که محل فعلی خراب شود، اجرا را به‌عهده می‌گیرند. به‌طور خاص‌تر، درحالی‌که عامل در حال اجرا شدن روی محل p_i در مرحلهٔ S_i است، اجرای آن توسط محل قبلی p_{i-1} کنترل می‌شود. به‌علاوه p_{i-1}، یک نسخه از عامل a_i را نگاه می‌دارد. درصورتی‌که در محل p_i خرابی رخ دهد، محل p_{i-1} نسخه‌ای از عامل را که در اختیار دارد مورد استفاده قرار می‌دهد و آن را به محل دیگری مانند p'_i می‌فرستد. با وجود فرستادن a_i به p'_i ممکن است به عامل‌های نسخه‌برداری‌شده بینجامد؛ برای مثال اگر که محل p_k به اشتباه، خرابی p_{k+1} را آشکار کند درحالی‌که p_{k+1} واقعاً خراب نشده باشد، در این حالت دو نسخه از عامل a_{k+1} اجرا شده و به دو عامل a_{k+2} و a'_{k+1} منجر می‌شود. عامل‌های نسخه‌برداری‌شده فقط در مقصد عامل آشکار می‌شوند و در آنجا تصمیم اجرا اتخاذ می‌شود. این کار، عملیات ویژهٔ فقط یک بار را ممکن می‌سازد. میزان تحمل‌پذیری خطا توسط تعداد نسخه‌هایی که روی محل‌هایی که عامل قبلاً روی آن‌ها اجرا و ذخیره شده است، تعیین می‌شود؛ به عبارت دیگر اگر عامل در لحظهٔ حاضر روی محل p_i اجرا شود، محل‌های $p_{i-1}, p_{i-2}, \ldots$ ممکن است به‌ترتیب نسخه‌های $a_{i-1}, a_{i-2}, \ldots$ از عامل را در خود ذخیره کنند. هرچه این تعداد بیشتر باشد، خرابی‌های همزمان بیشتری می‌توانند ترمیم شوند؛ باوجوداین، تعداد بیشتر نسخه‌ها، احتمال عامل‌های نسخه‌برداری‌شده را نیز افزایش می‌دهد. به هنگام ترمیم یک عامل خراب‌شده، لازم است که دو حالت زیر از یکدیگر متمایز شوند:

۱. عامل فقط به‌صورت جزئی روی آن محل اجرا شده است؛

۲. تمام عملیات مرحله‌ای را اجرا کرده و عامل را به محل بعدی فرستاده است.

در حالت اول، عامل ترمیم‌شده می‌تواند اجرای جزئی عملیات مرحله‌ای را به پایان برساند یا خنثی کند ولی در حالت دوم پیچیده‌تر است، زیرا عامل نمی‌داند که آیا ارسال موفق بوده است یا خیر.

۳.۴. مقایسهٔ رویکرد CAD و رویکرد CAS

مهم‌ترین تفاوت بین رویکرد CAD و رویکرد CAS، طول عمر عامل‌های نسخه‌برداری‌شده و تعداد تصمیمات اجرا می‌باشد. مسئلهٔ طول عمر، روی مدت زمانی که اجزای اطلاعاتی باید قفل‌شده بمانند، تأثیر می‌گذارد. هرچه این طول عمر بیشتر باشد، اجزای اطلاعاتی به مدت بیشتری قفل‌شده باقی می‌مانند. در این مدت، عامل‌های سیار دیگر نمی‌توانند به اجزای اطلاعاتی دسترسی پیدا کنند و مسدود می‌شوند، چراکه عملکرد کلی سیستم را محدود می‌کند. رویکرد CAS معمولاً عامل‌های نسخه‌برداری‌شده را در سطح مرحله آشکار می‌سازد، طول عمر عامل‌های نسخه‌برداری‌شده به اجرای مرحله، محدود می‌شود، برعکس رویکرد CAS در مقصد عامل باید قفل‌ها را روی اجزای اطلاعاتی تا پایان اجرای عامل نگه دارد. واضح است که این یکی از معایب رویکرد CAD می‌باشد. در مقصد عامل، تغییرات یک عامل اجرا می‌شوند، درحالی‌که تمام عامل‌های نسخه‌برداری آشکار شده و اثر آن‌ها خنثی می‌شود. خنثی‌سازی و اجرای عملیات‌های مرحله‌ای عامل، فرستادن پیام‌های بیشتری به تمام محل‌های مسیر حرکت را می‌طلبد. یکی دیگر از معایب

1. Single – Single – distributed
2. Multiple – Single - distributed

رویکردهای CAD، نیاز به ذخیره کردن نسخه‌هایی از حالت و کد عامل در موقعیت‌های چندگانه است. این کار مقدار حجم حافظهٔ قابل‌توجهی را می‌طلبد، درحالی‌که عامل‌های سیار عموماً اندازهٔ کوچکی دارند. به‌طور کلی، نسخه‌های عامل سیار باید تا زمانی که اجرای عامل به پایان برسد حفظ شوند، یعنی تا زمانی که پیام اجرا یا انصراف دریافت شده باشد. در رویکرد CAS، نسخه‌های عامل روی محل‌های M_i فقط در طول مرحلهٔ S_i ذخیره می‌شوند و بعد از آن کنار گذاشته می‌شوند. از سوی دیگر، رویکرد CAD عامل، با در نظر گرفتن تعداد تصمیمات اجرا، کارآمدتر است، درحالی‌که رویکرد CAD نیاز به یک تصمیم اجرا دارد، رویکرد، نیاز به n-2 تصمیم دارد؛ به عبارت دیگر، در رویکرد CAD در تمام مراحل (به جزء S_o و S_n) یک تصمیم اتخاذ می‌شود. علاوه‌براین، نسخه‌های عامل تنها در صورتی به‌کار گرفته می‌شوند که یک خرابی آشکار شود. برعکس، رویکردهای CAD، از هر راه‌حل MXX استفاده کنند، نسخه‌های عامل‌ها را به مرحلهٔ بعد می‌فرستند اگرچه ممکن است، هیچ خرابی‌ای آشکار نشده باشد.

۵. رویکردهای اجرای عامل سیار تحمل‌پذیر خطا

در این بخش، یک بررسی از رویکردهای موجود برای عامل‌های سیار تحمل‌پذیر خطا ارائه می‌شود. ابتدا رویکردهای CAS ارائه می‌شوند سپس رویکردهای CAD مورد بررسی قرار می‌گیرند. نتایج این دسته‌بندی در جدول (۱) به‌طور خلاصه آورده شده است.

جدول (۱): رویکردهای موجود برای عامل‌های سیار تحمل‌پذیر خطا

CAD	CAS	رویکردها عامل سیار تحمل‌پذیر خطا
____	روش Vogler [۳۱-۳۲]	SSC
____	روش Rothermel [۳۳]	MSC
____	روش‌های FATOMAS [۲۸]، NAP [۳۴] و Schneider [۳۵]	MMC
روش‌های James [۳۶] و Netpebbles [۳۷]	____	MSD
___	روش‌های James [۳۶]، FANOTOMAS [۳۸] و Assis [۳۹]	MMD

۵.۱. رویکردهای CAS

رویکرد خرابی‌های Byzantine: Minsky و همکارانش [۴۰] و *Schneider* [۳۵] اجراهای چند بار عامل سیار را به‌عنوان یک رویکرد اساسی برای فراهم آوردن عدم آسیب‌پذیری در مقابل خرابی‌های Byzantine، به‌خصوص در برابر حملات میزبان‌های بدخیم، روی عامل سیار پیشنهاد می‌کنند. در این زمینه، تمام، محل‌های $P_i^j \in M_i$ در مرحلهٔ S_i (محل‌های مرحله S_i)، عامل a_i را اجرا کرده و تغییرات را بر عهده می‌گیرند. اگر چه یک مورد مخالف ممکن است تعدادی از عامل‌ها را در یک مرحله خراب کند، اگر عامل‌های خراب‌نشده به تعداد کافی باقی مانده باشند، هنوز هم می‌توان نتیجهٔ واقعی اجرای مرحله را به‌دست آورد؛ بنابراین ویژگی اجرای فقط یک بار در این پروتکل، مطلوب نیست. در واقع، با اجرای عامل روی تمام محل‌ها، یک سطحی از پایداری ایجاد می‌شود. رویکرد *Schneider* را می‌توان به‌عنوان یک رویکرد MMC در نظر گرفت، که در آن محل‌ها همیشه به‌صورت یک‌جانبه تصمیم می‌گیرند تا تغییرات نسخه عامل را بر عهده بگیرند.

FANTOMAS[1]: رویکرد تحمل‌پذیر خطا برای عامل‌های سیار در [۳۸] یک رویکرد MMD است که به تحمل‌پذیری خطا برای کاربردهای توزیع‌شده و موازی در سیستم‌های گروهی می‌پردازد.

مکانیزم‌های تحمل‌پذیری خطای این رویکرد می‌توانند به هنگام درخواست عامل، فعال شوند. FANTOMAS فقط یک خرابی را در هر بار در نظر می‌گیرد. به همراه هر عامل، به‌اصطلاح یک عامل لاگر وجود دارد که به فاصلهٔ d عامل را دنبال می‌کند؛ برای مثال، اگر عامل روی محل p_i اجرا شود، عامل لاگر روی محل p_{i-2} قرار خواهد داشت. پس فاصلهٔ d برابر ۲ می‌باشد. عامل لاگر بازرسی نقطه‌ای، عاملی را که با آن همراه است، ذخیره می‌کند. برای رسیدن به این مقصود، عامل به‌طور متناوب حالت را به عامل لاگر می‌فرستد. عامل و عامل

1. Fault- Tolerant approach for Mobile Agents

لاگر، همدیگر را کنترل می‌کنند و به هنگام خرابی هریک از آن‌ها، آن دیگری می‌تواند از اطلاعات ذخیره‌شده در عامل سالم استفاده کرده و به حالت اول بازگردد. درصورتی‌که بیش از یک محل به‌طور همزمان خراب نشود، انسداد به‌وجود نمی‌آید. متأسفانه آشکار کردن نامطمئن خرابی ممکن است به نقض ویژگی اجرای فقط یک بار بینجامد. فرض کنید که عامل لاگر به خطا، خرابی عامل را آشکار و ترمیم کند. این کار به دو عامل و در نتیجه اجراهای چند بار کد عامل منجر می‌شود. بااین‌حال FANTOMAS به سیستم‌های گروهی می‌پردازد. رویکرد موجود در [۳۸] بسیار کارآمد بوده و می‌تواند بدون دخالت مالک عامل، به‌صورت پویا متوقف یا اجرا شود. به‌علاوه، مکانیزم‌های تحمل‌پذیری خطا برای مالک عامل، شناخته‌شده می‌باشند.

FATOMAS[1]: رویکردی که در مقالهٔ [۲۸] ارائه شده است، یک رویکرد MMC از CAS است. افزونگی نشان داده‌شده در این مقاله، اجرای عامل سیار را قادر می‌سازد تا علی‌رغم وجود خرابی‌ها ادامه یابد؛ یعنی از انسداد جلوگیری کرده و اجرای پایدار را نیز تضمین می‌کند. این مسئله به این واقعیت برمی‌گردد که الگوی سیستمی را در نظر می‌گیرد که آشکارسازی خرابی در آن نامطمئن است. راه‌حلی که ارائه می‌شود بایستی برای تمام نسخه‌های عامل در مرحلهٔ S_i، شامل حل یک مسئله موافقت باشد. در یک مسئلهٔ موافقت، روی یک مقدار مشترک، تصمیم‌گیری می‌شود.

JAMES: JAMES [۳۶] سیستمی است که به رویکرد MMD تعلق دارد. بااین‌حال، رویکرد CAD را نیز دربرمی‌گیرد.

NAP: الگوریتم NAP [۳۴] از رویکرد MMC برای اجرای عامل سیار تحمل‌پذیر خطا استفاده می‌کند. این رویکرد یک مدل خرابی- توقف را (که یک آشکار ساز خرابی کامل را تداعی می‌کند)، به کار برده و از انسداد به‌دلیل ماهیت رویکرد MMC جلوگیری می‌کند. درحالی‌که ویژگی اجرای فقط یک بار توسط فرض آشکارساز کامل خرابی تضمین می‌شود. NAP تنها در سیستم‌هایی که در آن‌ها آشکارسازی کامل خرابی می‌توانند فرض شوند، قابل استفاده است. علاوه‌براین رویکرد NAP خرابی‌های اتصال را حل نکرده و بازیابی محل‌ها را هم مورد توجه قرار نمی‌دهد.

رویکرد Rothermel: رویکرد Rothermel [۳۳]، مشابه MSC انسدادی است. خرابی یک محل اجرا، تصمیم اجرا را نیز مسدود می‌کند و درنتیجه، اجرای عامل سیار را نیز مسدود می‌کند. در [۳۳]، Rothermel و Strasser رویکردی براساس تعامل‌ها، و انتخاب رهبر پیشنهاد می‌کنند. مسئلهٔ انتخاب رهبر شامل انتخاب رهبر بین یک گروه از پروسه‌هاست، به‌طوری‌که فقط یک رهبر وجود دارد و همهٔ پروسه‌های این گروه روی آن رهبر موافقت می‌کنند. عامل بین دو محل متوالی S_i, S_{i+1} با استفاده از صفوف پیام تعاملی، به جلو هدایت می‌شود؛ به عبارت دیگر، یک محل p_i^j، عامل a_{i+1} را در صف پیام ورودی محل p_{i+1}^k به‌عنوان بخشی از یک تعامل کلی قرار می‌دهد. این تعامل کلی، مشابه اجرای مرحلهٔ کامل در S_i است و شامل موارد زیر می‌باشد:

۱. گرفتن عامل a_i از صف پیام ورودی؛

۲. اجرای عملیات مرحله‌ای عامل؛

۳. قرار دادن عامل به‌دست‌آمده a_{i+1} درون صف پیام محل‌های M_{i+1}. چندین محل در M_i به‌صورت بالقوه، این تعامل را اجرا می‌کنند، اما فقط رهبر عامل را اجرا کرده و تمام محل‌های دیگر عملیات مرحله‌ای عامل را متوقف می‌کنند. این رویکرد که با به‌کارگیری تعامل‌های محلی همراه است، اجرای فقط یک بار عامل را تضمین می‌کند، ولی متأسفانه در مقابل انسداد، آسیب‌پذیر است. علت این آسیب‌پذیری استفاده از یک پروتکل اجرای دومرحله‌ای، برای اجرای اتوماتیک تعامل‌هاست. پروتکل *2PC* در انسدادی بودن به‌علت یک خرابی، شناخته شده است [۴۱]. امکان دارد این نظر مطرح شود که استفاده از یک *3PC* مشکل انسداد را حل می‌کند، اما به‌دلیل آنکه متأسفانه انسداد از ترکیب انتخاب رهبر و تعامل‌ها و همچنین ماهیت نشئت می‌گیرد، با استفاده از *3PC* نمی‌توان از آن جلوگیری کرد.

رویکرد Assis: Assis [۳۹]، الگوریتم Rothermel را با غلبه بر برخی محدودیت‌های آن بهبود بخشیدند؛ به‌ویژه برای

1. Fault- Tolerant approach for Mobile Agents

جلوگیری از انسداد در [۳۳]، آن‌ها از یک پروتکل انتخاب رهبر متفاوت استفاده کردند و تعامل‌های محلی را با استفاده از یک پروتکل اجرای ۳ مرحله‌ای *3PC* بر عهده می‌گیرند. بااین‌حال، این ترکیب خاص انتخاب رهبر و مدل تعامل ممکن است به نقض ویژگی فقط یک بار بینجامد. بنابراین مقاله [۳۹]، بر یک بانک اطلاعاتی توزیع‌شده تکیه دارد، تا از وجود بیش از یک رهبر به‌طور همزمان جلوگیری کند و درنتیجه، ویژگی فقط یک بار را اعمال می‌کند. به‌طور خلاصه، تصمیم اجرا با همکاری بانک اطلاعاتی توزیع‌شده، یک پروتکل انتخاب رهبر و 3PC انجام می‌شود. به‌علاوه، بانک اطلاعاتی توزیع‌شده، روی محل‌های مرحلهٔ S_i در حال اجراست. همچنین این بانک اطلاعاتی می‌تواند به‌عنوان یک سرویس جدا مورد استفاده قرار گیرد.

درنتیجه، رویکرد استفاده‌شده در [۳۹] را به‌عنوان یک رویکرد MMD در نظر می‌گیریم. مشابه مقالهٔ [۳۳]، رویکرد [۳۹] از تعامل‌ها و انتخاب رهبر برای مدل‌سازی اجرای عامل سیارتحمل‌پذیر خطا استفاده می‌کند. عیب دیگر این رویکرد، هزینهٔ نسبتاً بالای نگهداری بانک اطلاعاتی توزیع‌شده است که لازم است (برای فراهم ساختن تحمل‌پذیری خطا) نسخه‌برداری شود.

رویکرد Vogler: *Vogler* و همکارانش [۳۱ـ۳۲] از رویکرد SSC استفاده کرده‌اند. تأکید اصلی آن‌ها روی تضمین اجرای فقط یک بار برای انتقال عامل بین دو محل متوالی P_i, P_{i+1} است. برای حصول این هدف، P_i تعاملی را آغاز می‌کند که شامل فرستادن عامل، ذخیره کردن عامل در P_{i+1}، به‌کار انداختن عامل در P_{i+1}، و حذف نسخهٔ عامل در محل P_i می‌شود. تا به اینجا آنچه مشخص است، این است که خرابی‌های عامل به هنگام اجرای عملیات مرحله‌ای در محل، مد نظر قرار نمی‌گیرند. بااین‌حال، این حقیقت که یک نسخه از عامل در مقصد ذخیره می‌شود، بازیابی خرابی محل را ممکن ساخته و اجازه می‌دهد تا تعامل محلی را دوباره از اول انجام شود. پرواضح است که رویکرد *Vogler* یک رویکرد انسدادی بوده، و ویژگی‌های اجرای فقط یک بار عامل سیار را به شرط اینکه عملیات مرحله‌ای عامل به‌صورت تعامل محلی انجام شود، تضمین می‌کند.

۲.۵. رویکردهای CAD

مهم‌ترین نمونه از رویکردهای CAD، سیستمی به نام Netpebble است [۳۷] که به گروه رویکردهای اجرای در مقصد MSD تعلق دارد. سیستم Jams را نیز می‌توان در این دسته‌بندی قرار داد، اگرچه برخی از عناصر رویکرد اجرای پس از مرحلهٔ MMD را شامل می‌شوند.

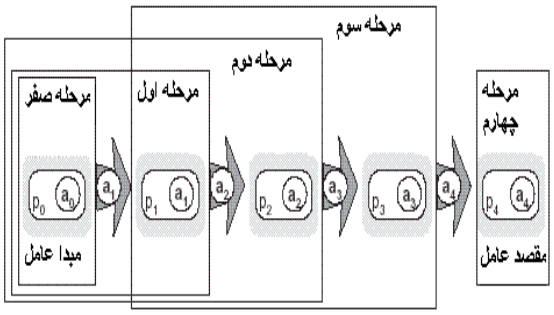
Netpebbles: محیط Netpebbles، عامل را به‌عنوان یک متن تعریف می‌کند که در بین محل‌ها حرکت می‌کند. این متن، حاوی کدی است که باید در هر محل اجرا شود [۳۷].

JAMES: JAMES [۳۶]، مدیران عاملی را تعریف می‌کند که به‌عنوان منبع عامل عمل می‌کنند و مدیریت و کنترل عامل‌های در حال اجرا را ممکن می‌سازند. این کار حمایت تحمل‌پذیری خطا را برای عامل‌های سیار فراهم می‌آورد، اما اجرای فقط یک بار عامل را تضمین نمی‌کند بلکه از مفاهیم اجرای حداکثر یک بار و حداقل یک بار استفاده می‌کند. این مفاهیم از ویژگی فقط یک بار ضعیف‌ترند. علاوه‌براین عامل سیار یا روی تمام محل‌های مسیر حرکتش یا روی ماکزیمم محل ممکن اجرا می‌شود. این مفاهیم اجرا به نظر می‌رسد که ابعاد مدیریت شبکهٔ مورد نظر در [۳۶] را مد نظر قرار می‌دهند، اگرچه هیچ مثال روشنی ارائه نشده است.

۶. انتشار مطمئن

روش ما هزینهٔ کلی را تا حد زیادی کاهش می‌دهد. برای رسیدن به این مقصود، فقط تعدادی از محل‌ها در هر مرحله، نتیجهٔ اجرای خود را به تمام محل‌ها در مجموعه محل‌ها در مرحلهٔ بعدی می‌فرستند. به‌منظور کاهش هزینه‌ها انتشار مطمئن را بهینه‌سازی می‌کنیم. به این صورت که هر مرحله در واقع یک محل دیگر را به مرحلهٔ قبلی اضافه می‌کند. در شکل (۵) همان‌طور که نشان داده می‌شود، برای مثال در مرحلهٔ دوم عامل a_2 روی محل P_2 و محل‌های P_0 و P_1 اجرا می‌شود. درصورتی‌که هیچ خرابی رخ ندهد، عامل روی محل P_2 اجرا

می‌شود، سپس عامل به محل‌های مرحلهٔ سوم فرستاده می‌شود.



شکل (۵): بهینه‌سازی انتشار مطمئن

از محل‌های موجود مرحلهٔ اول، عامل a_1 روی محل P_1 و محل P_0 اجرا می‌شود، که اگر خرابی رخ ندهد، عامل a_1 روی محل P_1 اجرا می‌شود و نتیجهٔ این اجرا به محل‌های مرحلهٔ دوم فرستاده می‌شود. در مرحلهٔ صفر و مرحلهٔ چهارم (مبدأ و مقصد عامل)، به‌دلیل آنکه هیچ نسخه‌برداری و افزونگی وجود ندارد، روال ذکرشده مورد نیاز نیست. در این حالت، هزینه‌های انتشار به کمترین حد رسیده و محدود به انتشار عامل به محل جدید می‌شوند (شکل ۵).

۱.۶. تفکیک مکانیزم‌های تحمل‌پذیری خطا

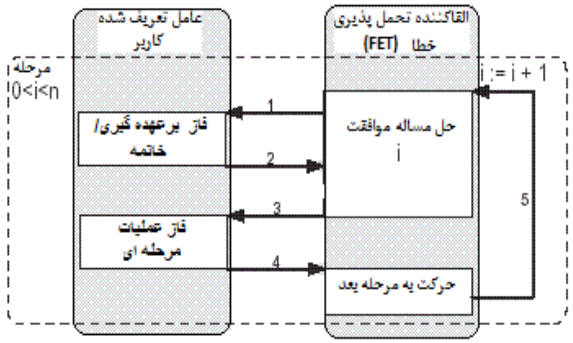
از لحاظ مفهومی، یک عامل سیار در سه مرحله اجرا می‌شود:

۱. فاز آغازین؛

۲. مراحل عملیات مرحله‌ای؛

۳. مرحلهٔ اتمام.

مرحلهٔ شروع روی مبدأ عامل (S_0) اتفاق می‌افتد، درحالی‌که مرحلهٔ اتمام (پایان) روی مقصد عامل (S_n) اجرا می‌شود. بین مبدأ و مقصد عامل، فاز عملیات مرحله‌ای در هر مرحله S_i $(0 < i \leq n)$ انجام می‌شود؛ بنابراین عامل چند بار اجرا می‌شود. به‌طور ایدئال، تحمل‌پذیری خطا باید بر عامل‌های سیار استوار بوده و مکانیزم‌های آن برای مالک عامل شفاف و روشن باشند. متأسفانه به‌دست‌آوردن شفافیت کامل مشکل است و عامل تعریف‌شده توسط کاربر، یعنی قسمتی که عملیات خاص کاربردی عامل را تعریف می‌کند، لازم است تا با مکانیزم‌های تحمل‌پذیری خطا تعامل داشته باشد؛ برای مثال درحالی‌که در اجرای یک عامل، آن عامل باید محل بعدی را که به آن حرکت خواهد کرد مشخص کند، اجرای عامل سیار تحمل‌پذیری خطای ما، عموماً یک مجموعه از محل‌های مقصد را برای مرحلهٔ بعدی (M_{i+1}) می‌طلبد. واضح است عامل از همانندسازی آگاهی دارد و شفافیت کامل دیگر امکان‌پذیر نیست. علاوه‌براین، اطمینان از تحمل‌پذیری خطا، یک فاز دیگر را به اجرای عامل می‌افزاید. همان‌گونه که اشاره شد، ردیابی ناقص خرابی ممکن است به نقض ویژگی فقط یک بار اجرای عامل سیار بینجامد. حل یک مسئله موافقت از اجراهای چندبارهٔ عامل از طریق تعیین کردن یک محل اصلی، جلوگیری می‌کند. تنها محل اصلی، عملیات را بر عهده می‌گیرد، درحالی‌که تمام محل‌های دیگری که عامل را اجرا کرده‌اند، باید عملیات عامل را خاتمه دهند؛ بنابراین به یک فاز بر عهده‌گیری/ خاتمه دادن نیازمندیم. برای مثال، تراکنش‌های بانک اطلاعاتی باید یا بر عهده گرفته شوند یا خاتمه پیدا کنند، درحالی‌که این عملیات عموماً نیازی به عمل بیشتر ندارند. در [۲۷ـ۲۸] یک معماری پیشنهاد شده است که مکانیزم‌های تحمل‌پذیری خطا را در جزئی به نام الحاقکنندهٔ تحمل‌پذیری خطا[1] (FTE) تفکیک می‌کند. شکل (۶) جریان تعامل بین FTE و عامل تعریف‌شده به‌وسیلهٔ کاربر را نشان می‌دهد. این تعامل در طول عملیات روی می‌دهد.



شکل (۶): جریان تعامل بین FTE و عامل تعریف‌شده به‌وسیلهٔ کاربر نشان داده می‌شود [۲۷ـ۲۸]

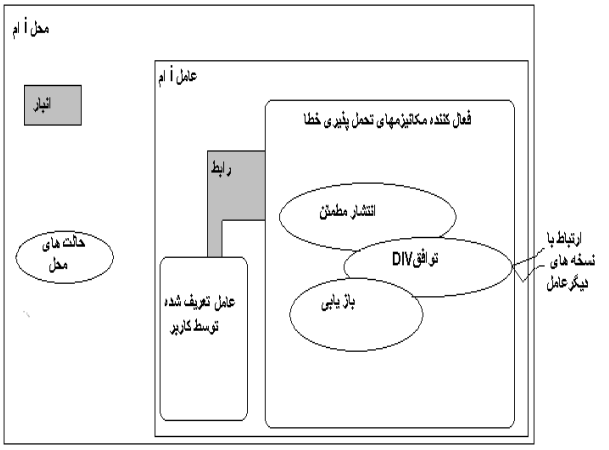
FTE مکانیزم های تحمل‌پذیری خطا را به‌صورت گروه جمع‌آوری می‌کند، درحالی‌که عامل تعریف‌شده توسط کاربر،

1. Fault Tolerance Enabler

دارای قسمت خاص کاربردی است. در هر مرحله S_i $(0 < i < n)$، FTE مشکل موافقت آن مرحله را حل می‌کند. عملیات انجام‌شده در فاز عملیات مرحله (فلش‌های ۱ و ۲ شکل ۶)، وابسته به نتیجهٔ موافقت، یا بر عهده گرفته می‌شوند و یا خاتمه می‌یابند (فلش‌های ۳و۴). سرانجام FTE عامل را به مجموعه محل‌ها در M_{i+1} می‌فرستد (فلش ۵)، که توسط عامل تعریفی کاربر محاسبه شده و به‌عنوان نتیجهٔ فاز عملیات مرحله‌ای برگشت داده می‌شود. دو رویکرد مربوط به محل قرار گرفتن FTE را می‌توان در نظر گرفت: رویکرد عامل‌محور (FTE با عامل) و رویکرد محل محور (FTE با محل‌ها).

۷. رویکرد وابسته به عامل

شکل (۷) معماری رویکرد عامل‌محور (وابسته به عامل) را نشان می‌دهد.



شکل (۷): ساختار رویکرد عامل‌محور

در ساختار این سیستم، مکانیزم‌های تحمل‌پذیر خطا در قسمتی به نام فعال‌کنندهٔ مکانیزم‌های تحمل‌پذیر خطا تفکیک می‌شوند، که خود این فعال‌کننده از سه جزء تشکیل می‌شود: جزء اول موافقت است که از توافق استفاده می‌کند (در هر مرحله یک مسئلهٔ موافقت را حل می‌کند و با توجه به نتیجهٔ این موافقت، اجرای مرحله‌ای عامل را بر عهده می‌گیرد یا آن را متوقف می‌کند). جزء دوم انتشار مطمئن است که وظیفهٔ انتشار عامل به مرحلهٔ بعد را بر عهده دارد و بالاخره جزء سوم جزء بازیابی است؛ زمانی که عامل خراب می‌شود یا با تأخیر به محل می‌رسد، عمل بازیابی توسط این جزء انجام می‌شود. از طریق یک رابط قسمت فعال‌کنندهٔ مکانیزم‌های تحمل‌پذیری خطا با عامل تعریف‌شده توسط کاربر به تعامل می‌پردازد؛ حاصل این تعامل یک عامل سیار تحمل‌پذیر خطاست. قسمت انبار در واقع محلی است که اطلاعات تحمل‌پذیری خطای محل، می‌توانند موقتاً در آن ذخیره شوند. در قسمت حالت‌های محل، خدماتی که یک محل برای عامل ارائه می‌دهد، قرار می‌گیرند.

۸. ارزیابی عملکرد و شبیه‌سازی‌ها

اندازهٔ فایل‌های دسته‌بندی‌شده برای یک عامل، حدود 8 kbyte و برای عامل همانندسازی‌شده 50 kbyte (که شامل دسته‌هایی برای FTE می‌شود) است. آزمایش‌های عملکرد روی ۷ ماشین AIX (قدرت PC :MHZ ۲۳۳ پردازشگر، رم (RAM) ۲۵۶ مگابایتی) انجام شده است. این ماشین‌ها به‌وسیلهٔ اینترنت 100Mbps به همدیگر متصل‌اند؛ ماشین‌ها روی ۳ زیرشبکهٔ مختلف قرار دارند.

هزینه‌های همانندسازی: دو مورد از هزینه‌های همانندسازی اندازه‌گیری‌شده عبارت‌اند از:

۱. هزینهٔ کلی مکانیزم همانندسازی، با در نظر گرفتن همانندسازی درجه‌یک؛

۲. هزینه‌های همانندسازی درجه ۳ (درجهٔ همانندسازی نشان‌دهندهٔ تعداد محل‌ها در یک مرحله و همچنین نشان‌دهندهٔ تعداد خرابی‌هایی است که الگوریتم درجه‌یک مرحله تحمل می‌کند).

همانندسازی درجه ۳، یک خرابی را حل کرده، و همانندسازی درجه ۵، دو خرابی را حل خواهد کرد. نتایج این اندازه‌گیری‌ها در جدول (۲) نشان داده شده است. این جدول میانگین حسابی ۱۰ اجرا را نمایش می‌دهد. ضریب ثابت تغییرات در بیشتر حالات بسیار کمتر از ٪۵ است. بااین‌حال، در مورد نتایج معدودی، این ضریب به ٪۱۵ می‌رسد.

ردیف اول در جدول (۲) هزینه‌ها را برای یک عامل منفرد نشان می‌دهد. عامل FTE (ردیف دوم در جدول ۱) از کد عامل

همانندسازی‌شده برای اجرا به شیوهٔ تک‌عامل استفاده می‌کند. در مقایسه با ردیف اول، ردیف دوم هزینهٔ کلی مکانیزم همانندسازی را نشان می‌دهد (هزینه‌های ارتباط و زمان محاسبه افزایش یافته است).

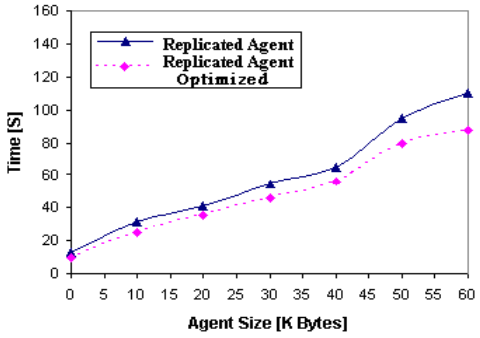
نتایج نشان می‌دهد که مکانیزم‌های همانندسازی حدود ٪۲۱ در مقایسه با مکانیزم‌های تک‌عاملی به هزینهٔ کلی می‌افزایند. هزینهٔ کلی در حالت سه‌مرحله‌ای، پایین‌تر بوده که در این مرحله ٪۱۶ است. ازآنجاکه هیچ ارتباطی بین مراحل میانی روی نمی‌دهد، یک عامل همانندسازی شده (که می‌تواند یک خرابی در هر مرحله را تحمل کند)، ۳ تا ۴ برابر پر هزینه‌تر از یک عامل منفرد است (ردیف سوم از جدول ۲). افزایش زمان اجرا به‌علت هزینه‌های اضافی ارتباط هدایت به جلو عامل و توافق است. در واقع بایستی اجرای یک عامل را در ۴ مرحله در نظر گرفت: اینجا ۳ پیام در مسیر بحرانی موجودند. از سوی دیگر با همانندسازی، ۱۲ پیام در مسیر بحرانی در مطلوب‌ترین طرح وجود دارد.

جدول (۲): هزینهٔ کلی مکانیزم همانندسازی، با در نظر گرفتن همانندسازی درجه‌یک و هزینه‌های همانندسازی درجه ۳ .

نوع عامل	۳ مرحله‌ای		۴ مرحله‌ای		۵ مرحله‌ای	
منفرد – ۴۰۰ کیلوبایتی	۶۲۱	۱۰۰ درصد	۱۲۶۵	۱۰۰ درصد	۱۶۷۵	۱۰۰ درصد
منفرد- ۴۰۰ کیلوبایتی با FTE	۸۴۳	۱۲۲ درصد	۱۶۰۲	۱۰۰ درصد	۲۰۹۸	۱۰۰ درصد
همانندسازی شده- ۴۰۰ کیلوبایتی با FTE	۲۷۰۳	۲۷۶ درصد	۳۸۷۶	۱۰۰ درصد	۵۶۷۸	۱۰۰ درصد
همانندسازی شده- ۴۰۰ کیلوبایتی با FTE - همراه با خرابی	۱۱۲۴۰	۱۲۵۶ درصد	۱۳۸۹۰	۱۰۰ درصد	۱۵۴۶۷	۱۰۰ درصد

علاوه‌براین در این آزمایش، زمان اجرای عملیات هر مرحلهٔ عامل کمتر از ۴ میلی‌ثانیه است؛ بنابراین مقدار قابل‌توجهی نیست. هرچقدر زمان اجرای عملیات مرحله‌ای عامل بیشتر باشد، نسبت هزینهٔ کلی بین تک‌عاملی و عامل همانندسازی‌شده کمتر خواهد بود. در نهایت، ردیف آخر در جدول (۲)، هزینه‌های اجرا را زمان خرابی نشان می‌دهد.

تأثیر اندازهٔ عامل: اندازهٔ عامل، تأثیر قابل‌توجهی روی عملکرد اجرای عامل سیار تحمل‌پذیر خطا دارد. برای اندازه‌گیری این تأثیر، عامل یک آرایش بایتی به طول‌های مختلف را با خود حمل می‌کند که برای افزایش اندازهٔ عامل به‌کار گرفته می‌شوند. همان‌گونه که نتایج در شکل (۸) نشان می‌دهد، زمان اجرای عامل به‌صورت خطی با افزایش اندازهٔ عامل افزایش می‌یابد. شکل (۸) نشان می‌دهد که هزینهٔ اجرای کامل عامل، ۱۳/۴ ثانیه است که نتیجه گرفته می‌شود که هزینهٔ کلی ارتباط حدود ۷ ثانیه است.

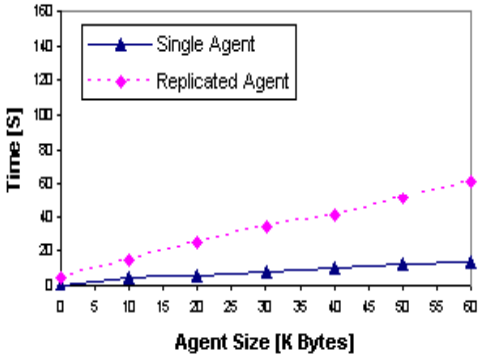


شکل (۸): زمان اجرای عامل که به‌صورت خطی با افزایش اندازهٔ عامل افزایش می‌یابد.

۸ـ۱. بهینه‌سازی

استفاده از شیوهٔ بهینه‌سازی‌شده منجر به کم شدن تعداد پیام‌ها می‌شود، به‌طوری‌که تنها لازم است که عامل منفرد، به یک محل جدید در مرحلهٔ بعد فرستاده شود. این کاهش پیام‌ها، در عملکرد کلی تأثیری نخواهد داشت، زیرا این الگوریتم تنها برای دریافت اولین پیام منتظر می‌ماند. بااین‌حال، کاهش تعداد پیام‌ها تأثیر قابل‌توجهی روی زیرساخت اصلی ارتباط خواهد داشت.

باوجوداین، شکل (۹) نشان می‌دهد که شیوهٔ بهینه‌سازی‌شده، زمان اجرای کمتری نسبت به عامل نرمال همانندسازی‌شده دارد. بازده عملکرد با افزایش اندازهٔ عامل افزایش می‌یابد؛ همان‌گونه که می‌توان انتظار داشت. در این آزمایش، از عاملی استفاده شده است که از ۸ مرحله دیدار کرده، که شامل مبدأ و مقصد عامل نیز می‌شود.

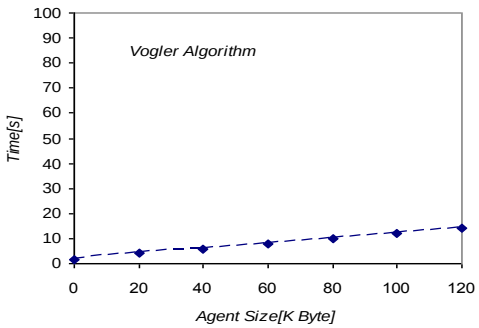


شکل (۹): شیوهٔ بهینه‌سازی‌شده، زمان اجرای کمتری نسبت به عامل معمولی همانندسازی شده دارد.

۸ـ۲. مقایسهٔ الگوریتم‌ها

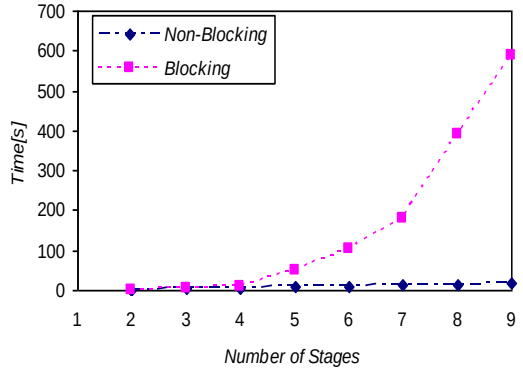
در این بخش، یک بررسی از الگوریتم‌های موجود برای عامل‌های سیار تحمل‌پذیر خطا ارائه می‌دهیم و در نهایت، آن‌ها را با یکدیگر مقایسه می‌کنیم.

الگوریتم Vogler: این الگوریتم، یک رویکرد CAS با راه‌حل SSC است که تأکید اصلی آن روی تضمین اجرای فقط یک بار، برای انتقال عامل بین دو محل متوالی است، و خرابی‌های عامل به هنگام اجرای عملیات مرحله‌ای در محل، مد نظر قرار نمی‌گیرد. در این الگوریتم، اجرای عامل توسط یک محل انجام شده و تصمیم اجرا نیز توسط همین محل اتخاذ می‌شود و اندازهٔ عامل، تأثیر قابل‌توجهی روی عملکرد اجرای عامل سیار تحمل‌پذیر خطا دارد. همان‌طور که در شکل (۱۰) مشاهده می‌شود، زمان اجرای عامل با افزایش اندازهٔ عامل به‌صورت خطی افزایش می‌یابد.



شکل (۱۰): نمایش زمان اجرای عامل با افزایش اندازهٔ عامل

در شکل (۱۰) که برای یک رویکرد CAS یا راه‌حل SSC است (الگوریتم Vogler)، هزینه برای کل اجرای عامل ۱۵/۲ ثانیه می‌باشد. پرواضح است که این الگوریتم یک الگوریتم انسدادی است. در شکل (۱۰) مقایسه‌ای از اجرای یک عامل غیر انسدادی و اجرای یک عامل انسدادی نشان داده شده است. همان‌گونه که بیان شد، در الگوریتم Vogler، تصمیم اجرا (CD) و اجرای عامل، هر دو در هر مرحله توسط یک محل صورت می‌گیرند.



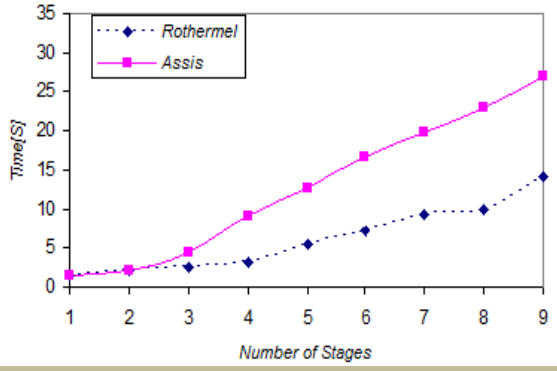
شکل (۱۱): مقایسهٔ زمان اجرای عامل انسدادی و غیر انسدادی با افزایش تعداد مراحل در الگوریتم Vogler

حال درصورتی‌که در این محل بر اثر وقوع یک خرابی، انسداد رخ دهد، اجرای عملیات مرحله‌ای عامل مسدود می‌شود و ادامهٔ اجرای عملیات مرحله‌ای عامل وابسته به ترمیم مجدد محل خراب‌شده است که فرایند ترمیم سبب افزایش زمان اجرای عامل خواهد شد. در شکل (۱۱) همان‌طور که نشان داده می‌شود، کل زمان اجرای یک عامل بدون خرابی ۳۲ ثانیه است و در زمانی که یک خرابی در مرحلهٔ ۵ رخ می‌دهد، زمان کل اجرا با احتساب زمان ترمیم به ۵۹۸ ثانیه می‌رسد. این الگوریتم، اجرای فقط یک بار عامل سیار را به شرط آنکه عملیات مرحله‌ای عامل به‌صورت تعامل محلی انجام شود، تضمین می‌کند. با توجه به مطالب فوق، این الگوریتم اغلب در محیط‌هایی استفاده می‌شود که در آن‌ها خرابی به‌ندرت رخ می‌دهد یا به عبارتی اصلاً دچار خرابی نمی‌شوند.

الگوریتم‌های Assis و Rothermel: در شکل (۱۲) مقایسه‌ای از زمان اجرای دو الگوریتم Rothermel و Assis در مراحل مختلف اجرای عامل نشان داده شده است. در

الگوریتم Rothermel، اجرای عامل فقط توسط یک محل انجام می‌شود، که این امکان را به‌وجود می‌آورد که خرابی در این محل به انسداد اجرای عامل سیار بینجامد.

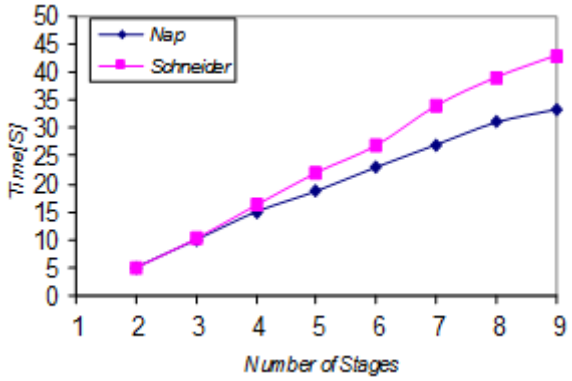
ولی تصمیم اجرا به‌دلیل توزیع شدن روی تمام محل‌ها غیر انسدادی است. اینکه تصمیم اجرا به‌صورت توزیع‌شده و اجرای عامل فقط بر روی یک محل رخ می‌دهد، عجیب به نظر می‌رسد.



شکل (۱۲): مقایسهٔ زمان اجرای عامل دو الگوریتم Rothermel و Assis با افزایش تعداد مراحل

بنابراین این الگوریتم به‌ندرت استفاده شده، و از آن در کاربردهای خاص استفاده می‌شود. اما در الگوریتم Assis به‌دلیل توزیع بودن تصمیم اجرا و اجرای عامل، از انسداد جلوگیری می‌شود. همان‌طور که در شکل (۱۲) مشاهده می‌شود، زمان اجرای عامل در الگوریتم Assis با افزایش تعداد مراحل به مراتب بیشتر از الگوریتم Rothermel است.

الگوریتم‌های Schneider و Nap: الگوریتم Nap از رویکرد CAS با راه‌حل MMC استفاده می‌کند. این الگوریتم بر آشکارسازی مطمئن خرابی تکیه دارد. از انسداد به‌دلیل ماهیت MMC جلوگیری می‌کند، درحالی‌که ویژگی اجرای فقط یک بار را توسط فرض آشکارسازی کامل خرابی، تضمین می‌کند. الگوریتم Nap تنها در سیستم‌هایی که در آن‌ها آشکارسازی کامل خرابی در نظر گرفته می‌شود، قابل استفاده است. علاوه‌برآن، این الگوریتم خرابی‌های اتصال را حل نکرده و ترمیم محل‌ها را در نظر نمی‌گیرد. با در نظر نگرفتن هیچ نوع بازیابی و ترمیم، در واقع یک منبع دیگر نقص ویژگی اجرای فقط یک بار حذف می‌شود. بااین‌حال، تعامل‌های محلی به‌منظور تضمین تفکیک عملیات‌های مرحله‌ای مورد نیازند. الگوریتم Schneider نیز همانند الگوریتم Nap از یک رویکرد CAS با راه‌حل MMC استفاده می‌کند. در این الگوریتم پس از آنکه یک جزء دچار خرابی می‌شود، قسمت‌های مختلف آن جزء، پاسخ‌های متفاوتی می‌دهند، که دستیابی به یک توافق در مورد تشخیص خطا به مسئلهٔ Byzantine منجر می‌شود. در این الگوریتم محل‌های یک مرحله همواره به‌صورت یک‌جانبه تصمیم می‌گیرند تا تغییرات نسخهٔ عامل را بر عهده بگیرند. شکل (۱۳) مقایسه‌ای از دو الگوریتم Schneider و Nap را نشان می‌دهد.



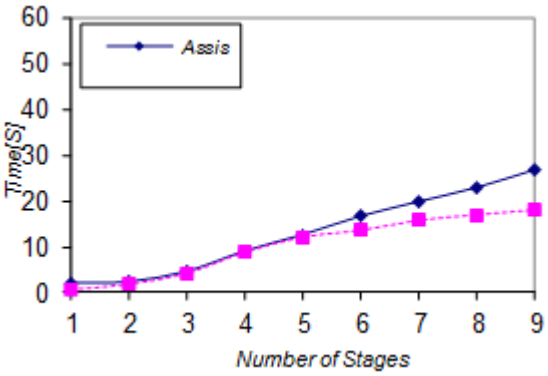
شکل (۱۳): مقایسه‌ای از زمان اجرای دو الگوریتم Schneider و Nap با افزایش تعداد مراحل

همانطور که در شکل (۱۳) مشاهده می‌شود، هزینهٔ اجرای عامل سیار در الگوریتم Nap نسبت به الگوریتم Schneider، با افزایش تعداد مراحل اجرا کمتر است. و علت اصلی آن، این است که در الگوریتم Nap همان‌طور که بیان شد، ترمیم و بازیابی محل‌ها و خرابی اتصال‌ها مد نظر قرار نمی‌گیرد. به همین دلیل زمان کل اجرای عامل در این الگوریتم نسبت به کل زمان اجرای عامل در الگوریتم Schneider که خرابی محل‌ها را در نظر می‌گیرد، کمتر است. الگوریتم Schneider، به‌دلیل آنکه اجرای عامل توسط تمام محل‌ها در هر مرحله انجام می‌شود و همچنین تصمیم اجرا توسط تمام محل‌های توزیع‌شده در هر مرحله اتخاذ می‌شود، از انسداد جلوگیری کرده و یک رویکرد غیر انسدادی است و اجرای فقط یک بار

را با استفاده از توافقی که درخصوص تشخیص خطا صورت می‌گیرد، تضمین می‌کند.

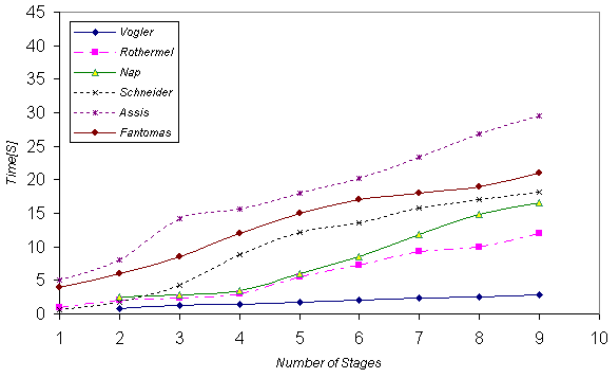
الگوریتم Fantomas: این الگوریتم مبتنی بر یک رویکرد CAS با راه‌حل MMD است که در رویکردهای تحمل‌پذیر خطا برای کاربردهای توزیع‌شده و موازی مورد استفاده قرار می‌گیرد.

این الگوریتم در هر مرحله، تنها یک خرابی را در نظر می‌گیرد. به همراه هر عامل، یک عامل Logger وجود دارد که به‌طور متناوب، عامل سیار حالات خود را به عامل Logger می‌فرستد. در واقع عامل و عامل Logger یکدیگر را کنترل می‌کنند و به هنگام خرابی هریک از آن‌ها، آن دیگری می‌تواند از اطلاعات ذخیره‌شده در عامل سالم استفاده کرده و به حالت اول بازگردد. به‌دلیل آنکه این الگوریتم و الگوریتم Assis هر دو از یک رویکرد و یک راه‌حل استفاده می‌کنند، در شکل (۱۴)، مقایسه‌ای از زمان اجرای این دو الگوریتم نشان داده شده است.



شکل (۱۴): مقایسه‌ای از زمان اجرای دو الگوریتم Assis و Fantomas با افزایش تعداد مراحل

در الگوریتم Fantomas درصورتی‌که بیش از یک محل به‌طور همزمان خراب شود، انسداد به‌وجود می‌آید، و آشکار کردن نامطمئن خرابی ممکن است به نقض ویژگی اجرای فقط یک بار بینجامد. باوجوداین، Fantomas در سیستم‌هایی که گمان اشتباه خرابی در آن‌ها بسیار نادر است، استفاده می‌شود. همان‌طور که در شکل (۱۴) مشاهده می‌شود، زمان اجرای عامل در الگوریتم Fantomas نسبت به الگوریتم Assis با افزایش تعداد مراحل کمتر است، و علت عمدهٔ آن به‌دلیل وجود عامل Logger برای هر عامل است. اجرای عامل در هر مرحله توسط تمام محل‌های آن مرحله صورت می‌گیرد و تصمیم اجرا نیز توسط محل‌هایی که از محل‌های اجراکنندهٔ عامل مجزا هستند، صورت می‌گیرد. در شکل (۱۵) مقایسه‌ای بین الگوریتم‌های بیان‌شده نشان داده می‌شود که در آن زمان، اجرای عملیات عامل با افزایش تعداد مراحل نشان داده شده است.



شکل (۱۵): مقایسه ای از الگوریتم‌های مبتنی بر رویکرد CAS با راه‌حل‌های متفاوت

همان‌طور که در شکل (۱۵) مشاهده می‌شود، الگوریتم Assis و سپس الگوریتم Fantomas به‌دلیل استفاده از راه‌حل MMD، با افزایش تعداد مراحل، زمان اجرای عامل در این الگوریتم‌ها بیشتر است. الگوریتم Schneider و سپس الگوریتم NAP که از راه‌حل MMC استفاده می‌کنند، نسبت به دو الگوریتمی که از راه‌حل MMD استفاده می‌کنند، به زمان کمتری برای اجرای عامل نیاز دارند. و الگوریتم Rothermel نسبت به ۴ الگوریتم فوق، به‌دلیل ماهیت MSC زمان اجرای عامل کمتری داشته، ولی به دلایلی که به آن‌ها در بخش‌های قبل اشاره شد، به‌ندرت و آن‌هم برای کاربردهای خاص استفاده می‌شود. در نهایت، در الگوریتم Vogler به‌دلیل آنکه هیچ نوع تکنیک نسخه‌برداری در آن به‌کار نمی‌رود، نسبت به تمام الگوریتم‌های بیان‌شده، زمان کمتری در اجرای عامل به‌ازای افزایش تعداد مراحل دارد. این الگوریتم به‌شدت

انسدادی بوده و تحمل‌پذیری خطای سیستمی را که از این الگوریتم استفاده می‌کند، کاهش می‌دهد.

۹. نتیجه‌گیری

در این مقاله، یک تقسیم‌بندی کلی از رویکردهای موجود برای اجرای عامل‌های سیارِ تحمل‌پذیرخطا ارائه شده است. متداول‌ترین نوع الگوی محاسباتِ کدسیار، عامل‌های سیارند که توجه زیادی را به خود جلب کرده‌اند. همچنین این مقاله به بررسی یک ساختار غیر انسدادی برای امنیت عامل‌های سیار می‌پردازد. برخلاف رویکرد مکان‌محور (وابسته به محل)، این ساختار از رویکردی وابسته به عامل (عامل‌محور) استفاده می‌کنند. در این رویکرد، پروتکلی که تحمل‌پذیری‌خطا را فراهم می‌سازد با عامل همراه بوده و به‌جای ترکیب مکانیزم‌های تحمل‌پذیری‌خطا با طرح عامل سیار، این مکانیزم‌ها نیز با عامل حرکت می‌کنند. این چارچوب کاملاً بر روی طرح عامل سیار voyager ساخته می‌شود؛ بنابراین عامل‌های سیار حتی می‌توانند به محل‌هایی بروند که تحمل‌پذیری خطا را حمایت نمی‌کنند. مزایای این رویکرد، در مقایسه با رویکردهای وابسته به محل، بسیار بیشترند. نتایج حاصل، هزینه‌های تحمل‌پذیری خطا را در مقایسه با یک عامل همانندسازی‌نشده نشان می‌دهند. این هزینه‌ها به تعداد مراحل و اندازهٔ عامل بستگی داشته، که افزایش اندازهٔ عامل و تعداد مراحل، زمان اجرا را افزایش می‌دهند. این هزینه‌ها با در نظر گرفتن تضمین‌هایی که عامل‌های سیار تحمل‌پذیر خطا فراهم می‌کنند، معقول و منطقی‌اند. در جدول (۳) مقایسه‌ای از الگوریتم‌ها آورده شده است.

جدول (۳): مقایسهٔ الگوریتم‌ها

الگوریتم	نوع رویکرد	رویکردها تحمل‌پذیر خطا	توضیحات
الگوریتم **Vogler**	CAS	SSC	تأکید اصلی روی تضمین اجرای فقط یک ب،ار برای انتقال عامل بین دو محل متوالی است.
الگوریتم **Rothermel**	CAS	MSC	در الگوریتم Rothermel ، اجرای عامل فقط توسط یک محل انجام می‌شود، که این امکان را به‌وجود می‌آورد، که خرابی در این محل به انسداد اجرای عامل سیار بینجامد.
الگوریتم **Assis**	CAS	MMC	در الگوریتم Assis به‌دلیل توزیع بودن تصمیم اجرا و اجرای عامل، از انسداد جلوگیری می‌شود. زمان اجرای عامل در الگوریتم Assis با افزایش تعداد مراحل به مراتب بیشتر از الگوریتم Rothermel است.
الگوریتم **Schneider**	CAS	MMC	در این الگوریتم، محل‌های یک مرحله همواره به‌صورت یک‌جانبه تصمیم می‌گیرند تا تغییرات نسخهٔ عامل را بر عهده بگیرند. در این الگوریتم پس از آنکه یک جزء دچار خرابی می‌شود، قسمت‌های مختلف آن جزء، پاسخ‌های متفاوتی می‌دهند که دستیابی به یک توافق دربارهٔ تشخیص خطا به مسئله Byzantine منجر می‌شود.
الگوریتم **Nap**	CAS	MMC	این الگوریتم بر آشکارسازی مطمئن خرابی تکیه دارد. الگوریتم Nap تنها در سیستم‌هایی که در آن‌ها آشکارسازی کامل خرابی در نظر گرفته می‌شود، قابل استفاده است.
الگوریتم **Fantomas**	CAS	MMD	این الگوریتم در هر مرحله، تنها یک خرابی را در نظر می‌گیرد. به همراه هر عامل، یک عامل Logger وجود دارد، که به‌طور متناوب، عامل سیار حالات خود را به عامل Logger می‌فرستد.

مراجع

[1] Hamidi, H., Vafaei, A., "Evaluation of Fault Tolerance Mobile Agents", International Journal of Intelligent Information Technologies, Vol.5, NO.1, pp.43-60, 2009.

[2] Chess, D., Harrison, C.G., Kershenbaum, A., "Mobile Agents: Are They a Good Idea?" In G. Vigna, Editor, Mobile Agents and Security, Springer Verlag, LCNS 1419, pp. 25-47, 1998.

[3] Pleisch, S., Schiper, A., "Approaches for Fault-tolerant Mobile Agents". Technical Report rz 3333, IBM Research, 2001.

[4] Hamidi, H., Mohammadi, K., "Modeling Fault Tolerant and Secure Mobile Agent Execution in Distributed Systems", International Journal of Intelligent Information Technologies Vol.2, NO.1, pp.21-36, 2006.

[5] Hamidi, H., Mohammadi, K., "Modeling and Evaluation of Fault Tolerant Mobile Agents in Distributed Systems", Proc, of the 2th IEEE Conf. on Wireless & Optical Communications Networks (WOCN2005), pp.91-95, 2005.

[6] Hamidi, H., Vafaei, A., "Evaluation of Security and Fault-Tolerance in Mobile Agents", Proc. of the 5th IEEE Conf. on Wireless & Optical Communications Networks (WOCN2008), pp.1-9. 2008.

[7] Hamidi, H., Vafaei, A., Monadjemi, S. A., "Evaluation and Check pointing of Fault Tolerant Mobile Agents Execution in Distributed Systems", Journal of Networks, Vol. 5, NO. 7, pp.18-29, 2010.

[8] Breugst, M., Busse, I., Covaci, S., Magedanz, T., "A Mobile Agent Platform for in Based Service Environments", in proc. IEEE Intelligent Networks Workshop, Bordeaux, France, pp.279-290, 1998.

[9] Strasser, M., Baumann, J., Hohl, F., "Mole – a Java Based Mobile Agent System", In M. Muhlhauser, Editor, Special Issues in Object Oriented Programming, pp. 301-308, 1997.

[10] Osman, T., Bargiela, A., "FADI: A Fault-Tolerant Environment for Open Distributed Computing", Proc. Inst. Elec. Eng. Softw. Vol.147, No.3, pp. 91-99, 2000.

[11] Pleisch, S., Schiper, A., "Fault - Tolerant Mobile Agent Execution". Computers, IEEE Transactions on, Vol. 52, Issue: 2, pp. 209 -222, 2003.

[12] Zhao, X., Shi, L., "DC Voltage Balance Control Strategy for Medium Voltage Cascaded STATCOM Based on Distributed Control", Journal of Networks, Vol. 9, No.1, pp. 138-146, 2014.

[13]X.Gao and Y. Yang, D. Zhou, J. Zhang, C. Chai. "Tar, et Tracking Approximation Algorithms with Particle Filter Optimization and Fault-Tolerant Analysis in Wireless Sensor Networks", Journal of Networks, Vol. 7, No.5, pp. 832-837. 2012.

[14] Tong, X., Wu, W., Ren, Bi. , "Application of Voltage Source on Electronics Power System", Beijing: Mechanical Industry Press, pp. 196-294, 2012.

[15] Christopher, D., Townsend, T., "Optimization of Switching Losses and Capacitor Voltage Ripple Using Model Predictive Control of a Cascaded H-Bridge Multilevel StatCom", IEEE Trans on Power Electronics, Vol. 28, No. 7, pp. 3077-3086, 2013.

[16] Wang, G., Jiang, J., Qiao, S., Guo, L., "Research on the Multilevel STATCOM based on the H-bridge Cascaded", Journal of Networks, Vol. 9, No. 1, pp. 147-152, 2014.

[17] Diego, E., Wheeler, P., "A Cascade Multilevel Frequency Changing Converter for High-Power Applications", IEEE Trans on Industrial Electronics, Vol. 60, No. 6, pp. 2118-2129, 2013.

[18] Zheng, Z., Li, G., Wang, N., "A Microcomputer-Based Predictive Digital Current Programmed Control System for Three-phase PWM Rectifier", Journal of Computers, Vol. 6, No 9, pp. 1880-1885, 2011.

[19] Townsend, C., Terrence, J., Betz, R.E., "Multi Goal Heuristic Model Predictive Control Technique Applied to a Cascaded H-bridge StatCom", IEEE Trans on Power Electronics, Vol. 27, No. 3, pp. 1191-1199, 2012.

[20] Sabel, L.S., Marzullo, K., "Simulating Fail-stop in Asynchronous Distributed Systems", In Proc. of the

13th IEEE Symposium on Reliable Distributed Systems (SRDS'94), pp138-147, 1994.

[21] Strasser, M., Rothermel, K., "Reliability Concepts for Mobile Agents", Int.J.Coop. Inform. Syst., Vol.7, No.4, pp. 355-382, 1998.

[22] Hirai, K., Mori, K., "Autonomous Assurance Techniqu in Distributed Data base using Mobile Agent", Proc. of ISADS 2000, pp180-187, 2000.

[23] Zamo, C. P. L., Wang, D., Mori, K., "Fault Tolerance Technology for Autonomous Decentralized Database System", Proc. of the 22nd IEEE International Symposium on Reliable Distributed system (SRDS'03), pp. 21-31, 2003.

[24] Kim, H., Yeom, H., Park, T., "The Cost of Check Pointing, Logging and Recovery for the Mobile Agent Systems", Proc. of the IEEE 2002. Pacific Rim International Symposium on Dependable Computing (PRDC'02), pp. 95-98, 2002.

[25] Qi, H., Xu, Y., "Mobile Agent Based Collaborative Signal and Information Processing in Sensor Networks", IEEE, Vol 91, No.8, pp.41-53. 2003.

[26] Guerraouim, R., Schiper, A., "Software-Based Replication for Fault Tolerance", IEEE Computer, Vol.30, pp. 68-74, 1997.

[27] Pleisch, S., Schiper, A., "Modeling Fault-Tolerant Mobil Agent Execution as a Sequence of Agreement Problems", In Proc. Of 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00), pp. 11-20, 2000.

[28] Pleisch, S., Schiper, A., "FATOMAS: A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach", In Proc. of Int. Conference on Dependable Systems and Networks (DSN'01), pp. 215-224, 2001.

[29] Osman, T., Bargiela, A., "An Approach to Rollback Recovery of Collaborating Mobile Agents", IEEE. Transactions on systems, Man, and Cybernetics, Part C: Applications and Reviews, Vol. 34, No.1, pp.23-34, 2004.

[30] Siam, A., Ramdane, M., Zaïdi, S., "Fuzzy Organization of Self-Adaptive Agents Based on Software Components", International Journal of Intelligent Information Technologies, Vol.10, No.3, pp: 36-56, 2014.

[31] Vogler, H., Kunkelmann, T., Moschgath, M.L., "An Approach for Mobile Agent Security and Fault Folerance Using Distributed Transactions", In Proc. of Int. Conference on Parallel and Distibuted Systems (ICPADS'97), Seoul, Korea, 1997.

[32] Vogler, H., Kunkelmann, T., Moschgath, M.L., "Distributed Transaction Processing as a Reliabilite Concept for Mobile Agents", In Proc. 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), Tunisia, 1997.

[33] Strasser, M., Rothermel, K., "A Fault-tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents", In Proc. of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98), pp. 100-108, 1998.

[34] Johansen, D., Marzullo, K., Schneider, F.B., Jacobsen, K., Zagorodnov, D., "NAP: Practical Fault-Tolerance for Itinerant Computations", In Proc. of the 19th Int. Conference on distributed computing Systems (ICDCS'99), Austin, Texas. 1999.

[35] Schneider, F.B., "Towards Fault-Tolerant and Secure Agentry", In proc. of the 11th Int. Workshop on Distributed Algorithms, Saarbrucken, Germany, 1997.

[36] Silva, L.M., Batista, V., Silva, J.G. "Fault - tolerant Execution of Mobile Agents", In Proc. of Int. Conference on Dependable System and Networks (DSN, 00), pp. 135-143, 2000.

[37] Mohindra, A., Purakayastha, A., "Exploiting non-Determinism for Reliability of Mobile Agent Systems", In Proc. of Int. Conference on Dependable Systems and Networks (DSN'00), pp.144-153, 2000.

[38] Pals, H., Petri, S., Grewe, C., "FANTOMAS: Fault Tolerance for Mobile Agents in Clusters", In J.D.P.Rolim, editor, Proc. of IPDPS 2000 Workshop, LNCS 1800, Springer Verlag, pp. 1236-1247, 2000.

[39] Assis Silva, F., Popescu-Zeletin, R., "An Approach for Providing Mobile Agent Fault-Tolerance", Proc. 2nd Int workshop on Mobile Agents, MA'98, Stuttgart, Germany, 1998.

[40] Minsky, Y., Van Renesse, R., Schneider, F.B.,

[41] Osman, T., Bargiela, A., "Fault-Tolerance for Mobile Agent Systems and Applications", in Proc. 2000 Agent- Based simulation , Passau , Germany , pp.45-56, 2000.

Stoller, S.D., "Cryptographic Support for Fault-tolerant Distributed Computing", In Proc of 7th European Workshop on ACM SIGOPS, pp.109-114, 1996.