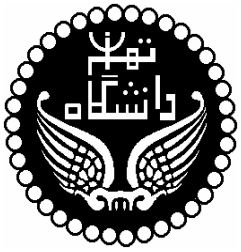
بسمه تعالي

دانشگاه تهران
دانشكدة فني
گروه برق وكامپيوتر



موضوع:

كمك‌رساني، روشي براي ايجاد تحمل‌پذيري‌خطا در سيستم‌هاي گسترده

نگارش :
مريم سادات ميريان حسين آبادي

استاد راهنما:
دكتر مجيد نيلي احمدآبادي
استاد مشاور:
دكتر زين‌العابدين نوابي شيرازي

پايان‌نامه جهت دريافت درجة كارشناسي ارشد در
رشتة مهندسي كامپيوتر
گرايش هوش ماشين و رباتيك
خرداد 1382

چکیده

در این تحقیق، روشهایی برای ایجاد تحمل‌پذیری خطا در سیستم‌های گسترده وبه ویژه سیستم‌های چندعامله ارائه شده است. ایجاد تحمل‌پذیری خطا در سیستم‌های گسترده، امری دستیافتنی و در عین حال دشوار است. یک سیستم چندعامله به عنوان نمونهٔ مهمی از سیستم‌های گسترده با بهره‌گیری از تکنیک‌های رایج تحمل‌پذیری خطا می‌تواند قابلیت کنارآمدن با انواع متفاوتی از خطاها را پیدا کند. اما آنچه در این تحقیق مورد تاکید است استفاده از ذات چندعاملهٔ سیستم‌ها و بهره‌گیری از تکنیک‌هایی است که برخاسته از ویژگی‌های خاص سیستم‌های چندعامله است.

در روشهای پیشنهادی حاصل از این تحقیق، در صورت بروز مشکلی برای یک یا تعدادی از عامل‌ها و اطلاع یافتن بقیه از این مطلب به وسیلهٔ صادر نمودن درخواست کمک، عامل‌های هم‌تیمی او با اعمال تغییراتی متناسب در وظایف خود سعی در کمک به او می‌نمایند و وظیفه وی را نیز بر دوش می‌گیرند و تا زمانی که مشکل آن عامل برطرف نشده، به این روش از کاهش کارایی سیستم جلوگیری می‌کنند. البته در صورت وجود چندین عامل نیازمند به کمک، عاملهای کمک‌رسان بر اساس الگوریتمی نسبتا بهینه که ایدهٔ اصلی آن شبیه اصول زمانبندی وظایف در سیستم‌های عامل است، اقدام به انتخاب وظیفه می‌کنند و بدون برقراری ارتباط صریح، مناسبترین وظیفه را از لحاظ سرعت و توان پردازشی(عملیاتی) و قابلیت اعتماد خود و نیز درجه اهمیت وظیفهٔ مورد نظر بر عهده می‌گیرند و از این طریق همکاری ضمنی میان عاملهای کمک‌رسان برقرار می‌شود. البته علاوه بر موارد ذکر شده، به وجود قابلیت های ویژه در عاملهایی که از این توانایی برخوردارند، نیز توجه شده است که این خود بر منطقی بودن تصمیم گیری در قبال تقسیم وظیفه ها می افزاید.

این همکاری با یک تصمیم‌گیری گسترده و طی چند فاز مختلف صورت می‌گیرد تا در صورت وجود چندین عامل نیازمند کمک، در نهایت عامل‌هایی با اولویت بالاتر برای کمک انتخاب شوند که بیش از بقیه در کارایی کلی تیم موثرند. در تصمیم گیری انجام شده، مهمترین هدف برآورده نمودن اهداف تحمل پذیری خطاست. بنابراین همواره سعی شده است که عاملهای بیشتری (ترجیحا با قابلیت ویژه) برای آینده به صورت رزرو باقی بمانند تا در صورت بروز هر خطای غیر منتظره، قادر به اقدام در جهت حفظ کارایی سیستم باشند.

گفتنی است که در این روشها، علاوه بر پرداختن به سیستم‌های قطعی که در آنها برای عاملها با پریود

معلومي وظيفه تخصيص مي‌يابد و عاملها تصميم‌هاي كاملا قطعي مي‌گيرند، روشهايي نيز براي كنارآمدن با عدم قطعيت پيشنهاد شده است و عاملها در چنين شرايطي قادر به تصميم‌گيري غيرقطعي و احتمالاتي بوده، بهترين تصميم را در مورد محتمل‌ترين شرايط اتخاذ مي‌نمايند.

روشهای پیشنهادی این تحقیق، بر روی یک بستر آزمون شبیه یک سیستم کنترل گسترده ساده آزمایش شده و با اعمال ایده‌های پیشنهادی در هر مرحله سعی در افزایش تحمل‌پذیری‌خطا در این سیستم کرده‌ایم.

اين روشها، توسط آزمايشهاي مختلفي شامل شرايط قطعي، غيرقطعي و نيز آزمونهاي تصادفي و آزمونهاي خاص با شرايط دشوارتر، بررسي شده‌اند و با توجه به معيارهاي ارزيابي پيشنهادي با يكديگر مقايسه شده‌اند. معيارهای ارزيابی علاوه بر يک معيار برای محاسبه كارايی، شامل معيارهايی برای شناخت زمان پاسخ متوسط سيستم، معياری برای شناخت كيفيت تقسيم وظيفه از نظر سازگاری كمک و كمک‌رسان و نيز ميزان منابع رزرو سيستم برای خطاهای آينده است كه معيار آخر منحصرا با توجه به هدف تحمل‌پذيری‌خطا تعريف شده است. نتايج آزمايشها حاكي از موثر بودن روشهاي ارائه شده در شرايط خطا در جلوگيري از كاهش چشمگير كارايي نسبت به حالت بدون كمک است. لازم به ذكر است كه روشهاي پيشنهادي به صورت كاملا توزيع شده و بدون نياز به سربار برقراري ارتباط مداوم ميان عاملها طراحي شده‌اند و اين خود يكي از محاسن عمدة روشهاي پيشنهادي است.

۱- مقدمه

در سال‌هاي اخير، باتوجه به جذابيت‌هايي كه در زمينة هوش‌ماشين ورباتيك وجود دارد تحقيقات زيادي در اين حوزه انجام شده است. بخش قابل‌توجهي ازاين تحقيقات درزمينة سيستم‌هاي گسترده است كه مورد توجه محققين شاخه‌هاي مختلف علوم كامپيوتر و مهندسين كنترل مي‌باشند.

به طور كلي اگر ذات يك وظيفه كه تنها از يك موجوديت مركزي و توانمند برمي‌آيد، قابل تقسيم ميان چند موجوديت كوچكتر و ترجيحا هوشمند و خودمختار باشد، يك سيستم چندرباته يا در حالت كلي‌تر چندعامله، جايگزين بسيار مناسبي است.

۱-۱- مزاياي استفاده از سيستم‌هاي گسترده

علاوه بر برتري‌هايي كه در حالت كلي براي چنين سيستم‌هايي مي‌توان برشمرد، دلايل زير به صورت خاص براي توجيه گسترش و استفاده روزافزون سيستم‌هاي چندرباته يا چندعامله، عنوان شده است: [۲۴]

۱) بسياري ازكاربردهاي رباتيكي عملا درمكان، زمان ياعمليات گسترده مي‌باشند وبنابراين نيازمند راه‌حلي متناسب هستند.

۲) بسياري ازماموريتها درصورت تقسيم‌شدن بين مولفه‌هاي همكار كه بصورت موازي عمل مي‌كنند بسيار سريعتر قابل انجام مي‌باشند.

۳) با فراهم آوردن افزونگي (در قابليتها يا تعداد عامل‌ها) مي‌توان قابليت اطمينان و تحمل‌پذيري خطا را افزايش داد.

٤) معمولاً ساخت تعدادي ربات يا عامل با توانايي اندك كه با همكاري يكديگر قادر به انجام يك مأموريت باشند، بسيار ارزان‌تر و عملي‌تر از ساخت يك ربات يا عامل است كه به تنهايي و با قابليت‌اطمينان مطلوب بتواند كل ماموريت را انجام دهد.

۱-۲- لزوم ايجاد تحمل‌پذيري خطادر سيستمهاي گسترده

با وجود پژوهش‌هاي فراواني كه دردانشگاهها و مراكزتحقيقاتي در مورد سيستم‌هاي چندعامله يا چندرباته انجام شده، هنوز در عمل و در بسياري ازكاربردهاي

دنياي‌واقعي، اين تكنولوژي جايگزين روشهاي موجود نشده است و همچنان سيستم‌هاي متمركز با تمامي انتقادهايي كه به آنها وارد است، در اغلب كاربردها ديده مي‌شوند. شايد دليل عمدة اين امر را بتوان در آسيب‌پذيري فراوان آنها نسبت به اشكال‌ها وخرابي‌هاي رايجي دانست كه در مكانيزم‌هاي مختلف سازندة رباتها رخ مي‌دهد، يا اينكه كنارنيامدن عاملها را با تغييرات فراواني كه در محيط عملياتي پوياي آنها رخ مي‌دهد، علت اين مساله دانست. هر يك از اين موارد يا دلايل مشابه اينها، لزوم افزودن‌قابليت تحمل‌پذيري‌خرابي به رباتها و عاملها و در نتيجه طراحي سيستم‌هاي چندعاملة با قوام[1] بيشتر را تقويت مي‌كند. در صورتي كه اين نيازمندي‌ها برآورده شود، با جايگزين نمودن سيستم‌هاي گسترده يا در حالت خاص، چندعامله به‌جاي سيستم‌هاي متمركز، مي‌توان تحمل‌پذيري خطا را بهبود بخشيد و در نتيجه كارايي سيستم را حتي با وجود خرابي در سيستم، در حد قابل‌قبولي باقي نگه داشت. تحقيق حاضر، به مسالة ايجاد تحمل‌پذيري خطا به كمك روشهاي ساده، عملي و در عين‌حال كم‌هزينه، در كاربردهايي مي‌پردازد كه قابليت توزيع‌شدگي[2] را دارند. منظور از كاربردهايي كه قابليت توزيع‌شدگي را دارند، كاربردهايي است كه هدف نهايي سيستم قابل تقسيم به چند زير هدف باشد و اين زير هدف‌ها هر يك به وسيلة موجوديت‌هاي مستقل با دانش‌هاي محدود، قابل اكتساب باشد. چنين سيستم‌هايي با بهره‌گيري از راهكارهايي كه در اين تحقيق معرفي شده‌اند، مي‌توانند طوري تغيير يابند كه حتي در صورت بروز خرابي عمده براي تعداد زيادي از عناصر سيستم تا جايي كه سيستم حداقل تعداد عناصر سالمي را شامل باشد، از كار باز نماند و حداقل بحراني‌ترين وظايفي كه در زمان طراحي برايش تعيين شده را به درستي به انجام برساند.

۱-۳- تحمل‌پذيري خطا در سيستم‌هاي چندعامله با روشهاي رايج

اگر توجه خود را در حالت كلي به سيستم‌هاي گسترده معطوف نماييم، هر سيستم گسترده را مي‌توان به منزلة تيمي از عامل‌هاي همكار[3] دانست كه هر يك وظيفة خاصي به عهده

1 Robustness
2 Distributed
3 Cooperative Agents

دارند و از تعامل آنها با يكديگر يك عملكرد واحد حاصل مي‌شود.

اين شيوة نگرش موجب مي‌شود كه بتوان از ايدة سيستم‌هاي چندعامله براي توسعة يك سيستم گستردة تحمل‌پذير خطا استفاده نمود. يكي از تكنيك‌هاي رايج براي ايجاد تحمل‌پذيري خطا در يك سيستم، استفاده از افزونگي[1] است. اما علاوه بر افزونگي صرف، در سيستم‌هاي چندعامله، تكنيك‌هاي پيشرفته‌تري هم براي ايجاد تحمل‌پذيري خطا وجود دارد و آن استفاده از خود عامل‌هاي موجود در سيستم براي رفع كاستي‌هاي ناشي از خطاي عامل هم‌تيمي آنهاست.

۱-٤- ايدة جديدي براي تحمل‌پذيري خطا در سيستم‌هاي چندعامله

در سيستم‌هاي چندعامله، بهره‌گيري صحيح از خاصيت چندعاملي بودن و عمل‌نمودن عامل‌ها به صورت مستقل، مي‌تواند نقطة اميدي باشد كه در صورت وقوع يك خطا، مشكل به كل سيستم توسعه نمي‌يابد.

بنابراين ايدة اساسي اين تحقيق آن است كه، به جاي جايگزين نمودن يك عامل خراب با يك عامل ديگر از خارج، از قابليت‌هاي در حال حاضر اضافة ديگر عاملهاي موجود در طرح استفاده كرده، نقش عامل خراب را نيز به نحوي بر دوش همكارانش قرار مي‌دهيم تا شرايط بحراني با كمترين تاثير نامطلوب بر كارايي مرتفع گردد. در اصل علت استفاده از يك محيط چندعامله براي بيان ايده‌هاي اين تحقيق آن است كه در چنين محيطي عناصر بالقوة ديگري براي تقبل نقش يك عامل خراب در سيستم وجود دارد و استفادة بهينه و منطقي از منابع موجود در سيستم، تحمل‌پذيري خطا را فراهم مي‌كند.

البته در هر سيستم گسترده مسالة تقسيم اولية وظايف[2] بايد با دقت انجام گيرد كه اين مي‌تواند يا به صورت دستي در زمان طراحي يا به صورت اتوماتيك در زمان اجرا توسط خود عاملها انجام شود. اگر مسالة انتخاب وظيفة اوليه را به عهدة خود عاملها بگذاريم، مي‌توان با مانيتورنمودن كارايي در زمان راه‌اندازي سيستم[3] توسط عاملها كه قابليت يادگيري دارند، انتظار داشت كه هر عامل مناسبترين كار را از لحاظ هماهنگي با ويژگي‌هاي خودش بر

1 Redundancy
2 Initial Task Assignment
3 Setup Time

عهده بگيرد. آنچه در اين تحقيق انجام شده است، تقسيم وظيفة دستي در زمان طراحي، ميان عاملهاست.

مسالة مهم ديگر كه در مسائل مشابه، گاه به آن پرداخته مي‌شود، مسالة مهم تقسيم يك وظيفه ميان چندعامل است. در صورتي كه يك وظيفه به علت خرابي يك عامل بر زمين مانده باشد و اين وظيفه قابليت تقسيم ميان چند عامل ديگر را داشته باشد، مي‌توان بخشهاي قابل انجام و مستقلي از آن وظيفه را به عاملهاي مختلف محول نمود و در آخر نتايج خروجي عاملها را با هم تركيب نمود تا يك نتيجة كامل به‌دست آيد. اما در اين تحقيق براي سادگي، وظيفه‌هايي براي عامل‌ها در نظر گرفته‌ايم كه به اصطلاح اتميك هستند و مستقلا بايد توسط يك عامل انجام گيرند ولي تمامي عاملها در صورت داشتن زمان كافي و دارا بودن توان پردازشي كافي براي تكميل آن قبل از تكميل مهلت زماني‌اش، قادر به انجام تمامي وظيفه‌ها هستند. البته بديهي است كه به علت دارا بودن قابليتهاي اعتماد مختلف احتمال تكميل يك وظيفه توسط هر عامل با عامل ديگر متفاوت است.

بنابراين در اين تحقيق، بهره‌گيري از خود عامل‌هاي درگير در حل مساله به منظور رفع خطا در سيستم مورد تاكيد است بدون اينكه از عامل واسطه‌اي[1] به منظور ايجاد هماهنگي در تنظيم وظايف جديد استفاده شود.

به‌طورخلاصه، يك عامل در صورت مواجهه با مشكلي كه وي را از ادامة همكاري بازدارد، يك پيام درخواست كمك صادر مي‌كند و ساير هم‌تيمي‌هاي خود را از اين واقعه مطلع مي‌كند. به بيان كلي‌تر، ديگر عاملها از خرابي هر يك از هم‌تيمي‌هاي خود آگاهي مي‌يابند. سپس از طريق فازهاي تصميم‌گيري گسترده كه اصل تاكيد اين تز است، اقدام به بررسي درخواست كمك و سپس انجام كمك‌رساني مي‌نمايند. لازم به توضيح است كه در اين تحقيق، به مسالة تشخيص خطا به شكل مستقل، توسط يك مولفة جداگانه پرداخته نشده است و فرض بر اين است كه خرابي يك عامل توسط خودش قابل تشخيص است. اين فرض براي سادگي انجام شده‌است، چرا كه پرداختن به روشهاي متعددي كه براي تشخيص خرابي وجود دارد، اين تحقيق را از مسالة اصلي خود كه كنارآمدن با خرابي در سيستم‌هاي گسترده است، منحرف مي‌نمود.

تا كنون از نتايج مقدماتي و مقطعي تحقيق حاضر، مقالات مختلفي به كنفرانسهاي بين‌المللي داخلي و خارجي

1 Middle Agents

ارسال و پذيرفته شده است كه [۳۹] اولين گام تحقيقاتي اين تز است، به بيان ساده‌ترين روش كمك‌رساني مي‌پردازد كه تنها مبتني بر درجه اهميت وظيفة عامل‌هاست.[٤٠] به بيان چند استراتژي سادة كمك‌رساني مبتني بر ريسك مي‌پردازد و در [٤١] مفهوم بي‌صبري براي كمك‌رساني با تعريف ساده‌اي معرفي شده است كه اين مفهوم در[٤٢] توسعه يافته و از دانش عاملها به نحو مناسبي استفاده مي‌نمايد. در[٤٣] استراتژي‌هاي ساده و تركيبي براي تعيين ترتيب كمك‌رساني معرفي شده‌اند. در[٤٤] روش غيرقطعي كمك‌رساني مبتني بر الگوريتم تقسيم وظيفة نسبتا بهينه مطرح شده‌است و [۴۵] شامل جمع‌بندي روشهاي مبتني بر تصميم‌گيري قطعي براي كمك‌رساني و تقسيم وظيفه است و بالاخره [٤٦] به جمع‌بندي تمامي روشهاي ارائه شده و مقايسة آنها مي‌پردازد.

ساختار اين گزارش به اين شكل است كه در فصل دوم، ادبيات حوزة تحمل‌پذيري خطا، تعاريف موجود در اين حوزه و روشهاي رايج ايجاد تحمل‌پذيري خطا بررسي شده‌اند. سپس در فصل سوم، پاره‌اي از پژوهش‌هايي كه تاكنون در رابطه با تحمل پذيري خرابي در سيستم‌هاي گسترده و چندعامله انجام شده، معرفي شده‌اند. در فصل چهارم، به دليل شباهت سيستم‌هاي چندعامله به سيستم‌هاي چندپردازنده‌اي و ايده‌اي كه در اين تحقيق از مسالة زمانبندي وظيفه‌ها در سيستم‌هاي عامل چندپردازه‌اي بلادرنگ گرفته شده‌است، چند الگوريتم تقسيم وظيفه به همراه مشكلات و مسائل خاصي كه استفاده از اين الگوريتم‌ها به همراه دارد، بررسي شده‌اند.

بعد از اين سه فصل و با معرفي اصولي كه من در ارائة روش خود از آنها بهره گرفته‌ام ، در فصل پنجم، سيستمي كه براي تحقق ايده‌هاي اين پروژه طراحي شده به همراه جزئيات كامل و فرضيات موجود شرح داده شده است. در ادامه، فصل ششم كه قسمت اصلي اين پايان‌نامه به حساب مي‌آيد، به ارائة روشهايي مي‌پردازد كه در اين تحقيق براي افزايش تحمل‌پذيري خطا در سيستم‌هاي چندعامله معرفي شده‌است. در فصل هفتم پس از معرفي چند معيار ارزيابي كارايي، با انجام آزمايش‌هاي متعدد كارايي روش‌هاي ارائه شده بررسي شده و صحت عملكرد آنها تاييد شده است. در فصل هشتم چگونگي عمليات سنتز و نتايج آزمايشهاي شبيه‌سازي پس از سنتز يكي از روشهاي پيشنهادي اين تز بر روي FPGA نشان داده شده است. و بالاخره در فصل نهم، نتايج تحقيق و پيشنهادهايي براي تكميل كار ارائه شده است.

۲- بررسي روشهاي رايج تحملپذيري خطا

امروزه با هر چه پيچيدهتر شدن سيستمهاي كامپيوتري، وجود عدم قطعيت در اغلب بخشهاي يك سيستم و لزوم دخالت عوامل متعدد در حصول خروجي، طراحي سيستمي كه با بروز مشكلي در هر يك از قسمتهاي آن، دچار نقص عمده نشده و قادر باشد عملكرد صحيح خود را حفظ نموده وصرفاً با تغييري در كارايي كلي ، هدف نهايي را برآورده سازد، بسيار ضروري است. با اين ترتيب ميتوان تعريفي از يك سيستم تحملپذير خطا به شرح زير ارائه نمود:

يك سيستم تحملپذير خطا، سيستمي است كه بتواند وظايفي كه برايش مشخص شده است، حتي با وجود خطاهاي نرم افزاري و خرابيهاي سختافزاري به درستي به انجام برساند. تحملپذيري خطا، به دليل مصرف رو به فزوني هر چه بيشتر كامپيوترها در كاربردهاي مختلف در زندگي بشر، بسيار مورد توجه قرار گرفته است و در واقع به علت اهميت كاربردها، تحملپذيري خطا يك ويژگي مهم در سيستمها محسوب ميشود.

۲-۱- اهداف تحملپذيري خطا

تحملپذيري خطا يك ويژگي مهم است كه به منظور دستيابي به برخي اهداف طراحي در سيستم منظور ميشود. درست همانطور كه يك طراحي بايد بسياري از اهداف و كاراييهاي مطلوب را برآورده سازد، بايد اهدافي از قبيل قابليت اطمينان[۱] ، دسترس پذيري[۲]، امنيت[۳] ، قابليت اجرا[۴] ، قابليت اعتماد[۵] ، قابليت نگهداري[۶] و آزمون پذيري[۷] را نيز فراهم نمايد.

۱Reliability

۲Availability

۳Safety

٤Performability

٥Dependability

٦Maintainability

۷Testability

۲-۱-۱- قابليت اطمينان

قابليت‌اطمينان $R(t)$، تابعي از زمان است كه يك احتمال شرطي را بيان مي‌كند، احتمال شرطي اين كه يك سيستم در طول مدت زمان (t_0,t_1) صحيح عمل نمايد، به شرط اينكه سيستم در زمان t_0، صحيح شروع بكار كرده باشد.قابليت اطمينان معمولاً در مورد سيستم‌هايي كه در آنها حتي پريودهاي زماني موقتي از عملكرد ناصحيح غيرقابل قبول باشد يا در آنها تعميركردن سيستم ممكن نباشد، بيشتر مورد توجه است.

توجه به تفاوت ميان تحمل پذيري خطا و قابليت اطمينان بسيار مهم است. تحمل‌پذيري خطا تكنيكي است كه مي‌تواند قابليت اطمينان را افزايش دهد ولي يك سيستم تحمل‌پذير خطا، لزوماً از قابليت اطمينان بالايي برخوردار نخواهد بود. به عبارتي مي‌توان سيستمي طراحي نمود كه هر خرابي سخت‌افزاري و خطاي نرم‌افزاري را تحمل نمايد اما احتمال رخ دادن اين موارد در سيستم آنقدر بالا باشد كه قابليت‌اطمينان را به حد بسيار پائيني برساند يا برعكس مي‌توان يك سيستم ساده را با استفاده از مولفه‌هاي بسيار خوب طراحي نمود، بطوري كه هر مولفه با احتمال بسيار پائيني دچار خطا شود ولي در صورتي كه اين اتفاق بيفتد، كار سيستم قطعاً متوقف شود. اين سيستم قابليت اطمينان زياد ولي تحمل‌پذيري خطاي كمي دارد.

۲-۱-۲- دسترس‌پذيري

دسترس‌پذيري، هدف ديگري براي طراحي است كه با استفاده از تحمل‌پذيري خطا بدست مي‌آيد. A(t) تابعي از زمان است كه به صورت يك احتمال شرطي تعريف مي‌شود؛ احتمال شرطي اينكه سيستم در لحظه t صحيح عمل نموده و در دسترس باشد.

تفاوت ميان دسترس‌پذيري و قابليت اطمينان در آن است كه قابليت‌اطمينان در يك بازه زماني تعريف مي‌شود در حالي كه دسترس‌پذيري در يك لحظه از زمان معنا دارد، بنابراين اگر يك سيستم به صورت پريوديك دچار خطا شود، باز هم ممكن است از دسترس پذيري خوبي برخوردار باشد چرا كه طول اين مدت زمانها مي‌تواند بسيار كوتاه باشد.

۲-۱-۳- امنيت

$S(t)$ اين احتمال را بيان مي‌كند كه يك سيستم كارش را به درستي انجام بدهديا به نحوي به كارش ادامه بدهد كه

ديگر سيستم‌هايي كه با آن در تماسند را دچار مشكل ننمايد و حتي سلامت افرادي كه در تعامل با آن هستند را با مخاطره مواجه ننمايد. امنيت معياري است براي اندازه‌گيري ميزان fail-Safe بودن يك سيستم ؛ يعني اگر سيستمي قادر نيست كه كارش را درست انجام بدهد، حداقل بتواند كه به شكلي امن، از كار بايستد.

تفاوت امنيت و قابليت‌اطمينان در اين است كه قابليت‌اطمينان احتمال صحيح انجام دادن كار سيستم را بيان مي‌كند در حالي كه امنيت يا بر عملكرد صحيح تاكيد دارد و يا در صورتي كه خرابي رخ بدهد، ترجيح بر آن است كه براي سلامت افراد خطري ايجاد نشود.

۲-۱-٤- قابليت اجرا

در بسياري از موارد، ممكن است كه سيستمي طراحي كنيم كه بتواند به عملكرد صحيح خود پس از رخداد خرابي سخت افزاري و خطاي نرم افزاري ادامه دهد اما سطح كارايي آن كمي دستخوش تغييرات شود. $P(l,t)$، تابعي از زمان و بيانگر احتمال اين مطلب است كه كارايي سيستم در زمان t مساوي يا بيش از l باشد.

مسأله كاهش قابل‌قبول كارايي[1] مسالة مهمي است كه در ارتباط تنگاتنگ با قابليت‌اطمينان مي‌باشد. تحمل‌پذيري خطا مي‌تواند كاهش قابل‌قبول كارايي را تامين نمايد، يعني با حذف سخت افزار يا نرم افزار خراب از سيستم و بنابراين ادامة عملكرد در سطح پائين‌تري از كارايي، اين امر را محقق نمايد.

۲-۱-٥- قابليت نگهداري

تقريباً كلية سيستم‌ها، اين مسأله را به عنوان هدفي در ليست اهداف مهم خود قرار مي‌دهند. قابليت نگهداري معياري است براي آنكه يك سيستم هنگامي كه با مشكلي مواجه شد به راحتي قابل تعمير باشد. به بيان رياضي‌تر، $M(t)$ احتمال آن است كه يك سيستم خراب بتواند در يك پريود مشخص از زمان t به وضعيت كارآمد بازگردد.

۲-۱-٦- آزمون پذيري

آزمون يا تست ، راهي است كه با كمك آن مي‌توان وجود و كيفيت برخي ويژگي ها را در يك سيستم مورد بررسي قرار

1 Graceful Degradation

داد و آزمون پذيري، توانايي آزمودن برخي خواص در يك سيستم است. آزمون پذيري و قابليت نگهداري با هم در ارتباطند ؛ چرا كه مينيمم نمودن زمان لازم براي تشخيص و تعيين موقعيت خطا از مسائل بسيار مهم است.

۲-۱-۷- قابليت‌اعتماد

واژة قابليت‌اعتماد، مفاهيم قابليت اطمينان ، دسترس‌پذيري ، امنيت ، قابليت نگهداري، قابليت اجرا و آزمون پذيري را شامل مي‌شود. قابليت اعتماد در اصل كيفيت سرويس دهي[۱] يك سيستم خاص را نشان مي‌دهد و اهداف ديگري از قبيل قابليت اطمينان، دسترس پذيري، امنيت و .... ، خود معيارهايي براي اندازه‌گيري ميزان قابليت اعتماد سيستم هستند.

۲-۲- كاربردهاي يك سيستم تحمل‌پذير خطا

كاربردهاي موجود براي سيستم‌هاي تحمل‌پذير خطا به چهار دسته عمده تقسيم مي‌شوند: كاربردهاي با طول عمر طولاني[۲] ، محاسبات بحراني[۳] ، تعميرات و نگهداري‌هاي دشوار[٤] و در نهايت كاربردهايي كه لازم است بسيار دسترس پذير[٥] باشند . هر يك از اين كاربردها، نيازهاي مختلف و در نتيجه تكنيك هاي متفاوتي را مي‌طلبند.

۲-۲-۱- كاربردهاي با طول عمر طولاني

مهم‌ترين مثال از اين نوع كاربرد، سفينه‌هاي فضايي بدون سرنشين است، مثلا سفينه فضايي pioneer10 كه درسال ۱۹۷۲ به فضا پرتاب شد يا Mariner، Explorer يا Voyager نمونه‌هاي جالبي از اين دست هستند. اين سفينه ها بايد در فضا براي مدت طولاني با صحت و سلامت كامل، كار مي‌كردند و از آنجا كه هزينه طراحي و ساخت و راه اندازي چنين سيستم‌هايي بسيار بالاست، عقلاني نيست كه خرابي‌هاي الكترونيكي در فضاهاي دور از دسترس اجازه بروز پيدا كنند. كاربردهاي نوعي مشابه، احتمال متوسط ۹۵% را براي مدت ده سال كه طول عمر مفيد آنهاست، پيشنهاد مي‌كنند. به عنوان يك مثال

۱Quality of service

۲Long-life

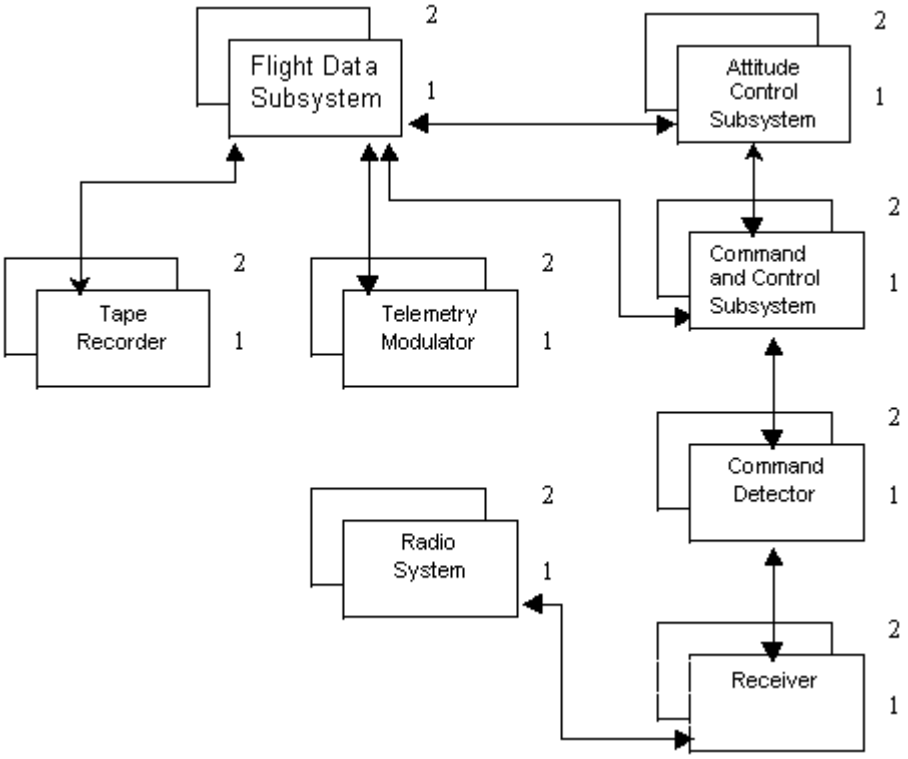
۳Critical computation

٤Maintenance postponement

٥Highly Available

از سيستم‌هاي كامپيوتري كه براي كاربردهاي با طول عمر طولاني مناسبند، يك دياگرام كلي از برد الكترونيكي سفينه فضايي Voyager را در شكل ١ مشاهده مي‌كنيد. سيستم شامل هشت مولفه اصلي است:

١) Flight Data System
٢) Attitude Control System
٣) Command and Control System
٤) Radio System
٥) Telemetry Modulator
٦) Command Detector
٧) Receiver
٨) Tape Recorder



شكل 1- برد الكترونيكي سفينه فضايي Voyager

در اين طرح، دو كپي مشابه از هر مولفه درنظر گرفته شده است:

يك كپي، كپي اصلي[١] ناميده مي‌شود و كلية عمليات را در حالت عادي انجام مي‌دهند، كپي دوم كه پشتيبان[١] نام

١Primary

دارد، هنگامي كه هر مولفه اصلي، با خرابي مواجه شود، فعال شده و جايگزين آن مي‌شود تا سيستم را در حالت فعال نگاه دارد.

۲-۲-۲- كاربردهاي با محاسبات بحراني

شايد مهم‌ترين كاربرد براي سيستم‌هاي تحمل‌پذيرخطا، كاربردهايي باشند كه در آنها عمليات محاسباتي در سلامتي انسان دخالت داشته باشد؛ مواردي از قبيل سيستم كنترلي هواپيما، سيستم‌هاي نظامي و برخي كنترلرهاي صنعتي.

يك كاربرد نوعي از اين دست، داراي قابليت اطميناني در حدود ۰/۹ در يك دوره تناوب سه ساعته است كه البته بسته به نوع كاربرد مي‌تواند كمي متفاوت باشد. سيستم‌هاي كنترل صنعتي مثل كنترل يك راكتور هسته‌اي بايد تا حدي با دقت انجام پذيرد كه هرگز انفجار ناخواسته‌اي رخ ندهد.

۲-۲-۳- كاربردهايي با تعمير و نگهداري دشوار

در اين نوع كاربردها هزينه نگهداري بسيار بالاست و به سختي انجام مي‌پذيرد. سيستم‌هاي پردازشي راه دور يا برخي كاربردهاي فضايي از اين دسته‌اند.

هدف اصلي استفاده از تحمل‌پذيري خطا، آن است كه عمليات تعمير و نگهداري به زمانهاي بعد كه احتمالاً امنيت بيشتري وجود خواهد داشت منتقل شود؛ مثلاً هر ماه يك بار پرسنل تعميرات به سيستم سركشي خواهند نمود و اگر مشكلي ايجاد شده باشد آنرا برطرف مي‌كنند؛ ولي در اين ميان سيستم بايد قادر باشد با خرابي‌هاي به وجودآمده كنار بيايد.

۲-۲-٤- كاربردهاي با دسترس پذيري بالا

دسترس پذيري به مرور به يك پارامتر كليدي در بسياري از كاربردها تبديل شده است. بانكداري و ديگر سيستم‌هاي اشتراك زماني مثال‌هاي مناسبي براي چنين سيستم‌هايي هستند. كاربران اين سيستم‌ها مايلند كه با احتمال بالايي در هر زمان كه از اين سيستم‌ها درخواست سرويس نمودند، سيستم قادر به دادن سرويس مورد نظر آنها باشد.

---

۱ Backup

۲-۳- تحمل‌پذيري خطا به عنوان يك هدف طراحي

نماي كلي فرآيند طراحي، در شكل 2 ديده مي‌شود. نيازهاي سيستم از قبيل قابليت اطمينان به دو طريق بدست آمده است:

۱- طراحي سيستم
۲- ارزيابي سيستم

طراحي سيستم شامل هر دو تكنيك اجتناب از خطا و تحمل‌پذيري خطا است. اجتناب از خطا با اين هدف انجام مي‌گيرد كه از خرابي‌هاي سخت افزاري و خطاهاي نرم افزاري در ابتداي طراحي اجتناب نمائيم. مثال‌هايي از اين دسته تكنيك‌ها عبارتند از : انتخاب مولفه‌هاي با كيفيت مناسب، اعمال قواعد طراحي و بررسي مجدد طرح به صورت متناوب.

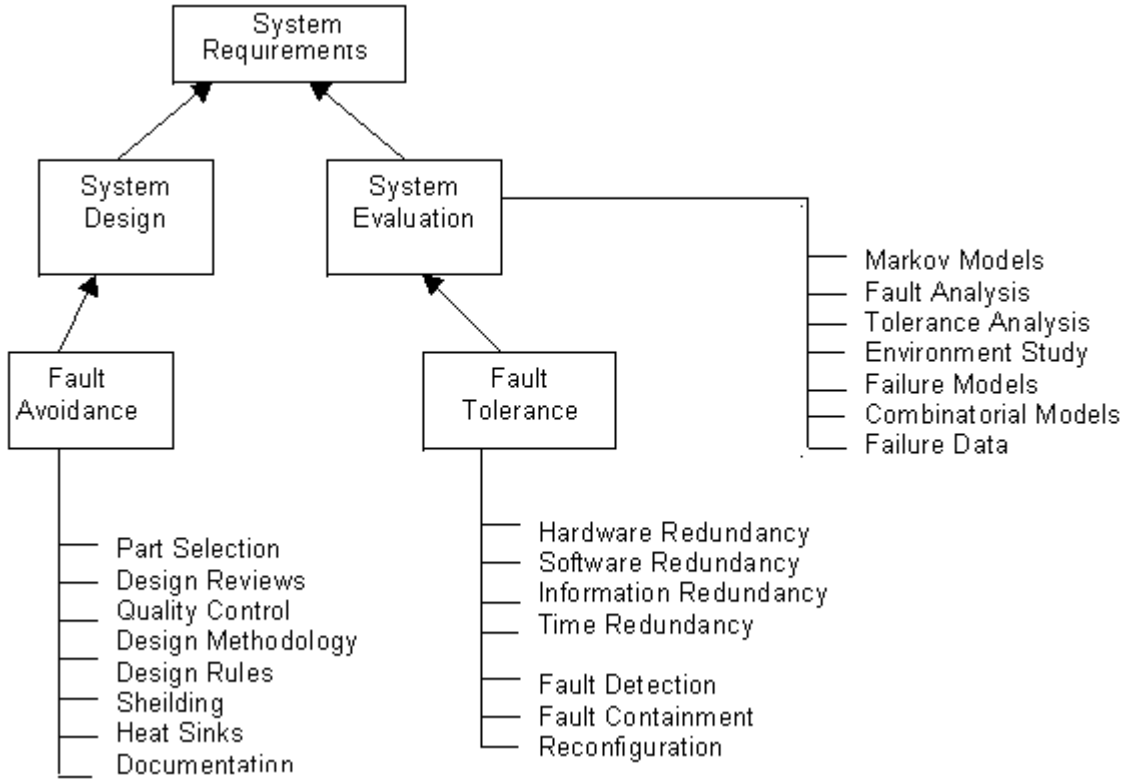
در اصل با اضافه نمودن سخت افزار افزونه[۱]، تكنيك‌هاي رأي‌گيري[۲] و سازماندهي مجدد[۳] ، مي‌توان به اهداف تحمل پذيري خطا دست يافت.

اگر بخواهيم كه فرآيند طراحي موفقيت آميز باشد، ارزيابي بايد به صورت موازي با فرآيند طراحي صورت پذيرد. براي مثال اگر يك مشكل طراحي پيش از آنكه طرح نهايي براي پياده سازي تهيه شود، مشخص گردد، معمولاً به راحتي قابل تغيير است.

۱Redundant Hardware

۲Voting

۳Reconfiguration



شكل 2- نمايش سطح بالاي فرآيند طراحي

روشهاي ارزيابي مختلفي براي آناليز سيستم‌ها وجود دارد؛ از قبيل مدل‌هاي قابليت اطمينان ماركف، مدل‌هاي تغيير سيستم، مدل‌هاي دسترس پذيري و مدل‌هاي قابليت نگهداري.

علاوه بر اينها، تكنيك‌هاي ارزيابي به ما اين امكان را مي‌دهند كه مكان‌هايي از سيستم را كه احتمال بروز خرابي در آنها وجود دارد را شناسايي كنيم.

۲-٤- تعاريف بنيادي

در دامنة صحبت از تحمل‌پذيري خطا، سه واژة Fault ، Failure و Error بايد از لحاظ تكنيكي از هم تفكيك شوند. ميان اين سه لغت در اين حوزه، رابطة علت و معلولي وجود دارد. به طور خاص ، Fault علت رخداد Error است و Error نيز موجب بروز Failure مي‌شود.

خطا[1] يك نقص فيزيكي يا عدم كاركرد يك مولفة سخت‌افزاري يا نرم‌افزاري است. به عنوان مثالي از خطا مي‌توان به اتصال كوتاه شدن دو سر يك رساناي الكتريكي يا به مدار باز شدن يك رسانا اشاره نمود. در نرم‌افزار

[1] Fault

منظور از خطا مي‌تواند وجود حلقه‌اي باشد كه پس از ورود به آن هيچگاه نتوان از آن خارج شد.

اشتباه[1] نتيجة وجود خطا در يك سيستم است. به طور خاص اشتباه يك انحراف از دقت يا صحت است. براي مثال در صورتي كه قسمتي از مدار دائماً اتصال كوتاه باشد، اين يك خطا است. اگر شرطي رخ بدهد كه لازم شود كه تغيير ۱ به ۰ انجام گيرد، مقدار آن خطا به اشتباه ۱ خواهد بود.

در آخر، اگر وجود اشتباه در سيستم موجب شود كه سيستم نتواند حداقل يكي از توانايي‌هاي خود را به درستي به انجام برساند، مي‌گوئيم كه در سيستم با خرابي[2] مواجه شده ايم.

مفاهيم Fault وError وFailure با استفاده از مدل سه جهاني زير به خوبي قابل بيان است:

دنياي اول، دنياي فيزيكي[3] است كه در آن خطا رخ مي‌دهد. اين دنيا شامل ابزار نيمه رسانا، عناصر مكانيكي، چاپگر، صفحه نمايش، منبع تغذيه و ديگر عناصر فيزيكي است كه سيستم از آنها ساخته شده است.

دنياي دوم، دنياي اطلاعات[4] است. اين جهان، همان جايي است كه در آن اشتباه رخ مي‌دهد. اشتباه هنگامي رخ مي‌دهد كه بعضي از واحدهاي اطلاعاتي با خرابي مواجه شوند.

دنياي سوم، دنياي خارجي[5] است كه گاه آنرا جهان كاربر هم مي‌ناميم. در اين جهان كاربر نتيجة خطا و اشتباه را مشاهده مي‌كند. اين جهان محل بروز خرابي است.

۲-۵- علل بروز خطا

رخدادن خطا مي‌تواند نتيجه عوامل متعددي باشد كه گاه در مرحله طراحي و گاه در فاز پياده‌سازي به آنها توجه نشده است. بسيار مهم است كه بتوانيم خطا را تشخيص بدهيم. يك فرآيند طراحي معمولاً با يك جمله شرح مسأله آغاز مي‌شود. سپس طراح براساس درك خودش از مسأله جزئيات بيشتري به شرح مسأله مي‌افزايد. با توجه به شرح مسأله يك راه حل سطح بالا ارائه مي‌شود و سپس توسعة الگوريتم‌ها و

۱Error

۲Failure

۳physical universe

٤Information universe

٥external universe

طراحي معماري آغاز مي‌گردد. هنگامي كه سخت‌افزار و نرم‌افزار تهيه شد، طراحي بايد از يك مرحله آزمايشي بصورت مجتمع عبور نمايد به طوري كه در نهايت عملكرد سيستم چنان باشد كه شرح اوليه مسأله را برآورده سازد.

مشكلاتي كه در هريك از فازهاي طراحي مي‌تواند بروز نمايد، مي‌تواند موجب بروز خطا در سيستم بشود. علل خطا هاي احتمالي مي‌تواند به هر يك از چهار مورد زير مرتبط باشد :

- توصيف اشتباه[1]
- پياده‌سازي اشتباه[2]
- خرابي عناصر
- عوامل خارجي[3]

مورد اول احتمال خرابي به علت اشتباه در توصيف است. اين مسأله به هر يك از موارد از قبيل طراحي الگوريتم اشتباه، معماري سخت‌افزار و نرم‌افزار اشتباه برمي‌گردد.

مورد بعدي اشتباهات پياده‌سازي است. پياده‌سازي در اصل فرآيند تبديل توصيف سخت‌افزار يا نرم‌افزار به سخت‌افزار واقعي و نرم‌افزار حقيقي است. به علت ضعف پياده‌سازي گاه ممكن است خطاهايي در اين مرحله ايجاد شود.

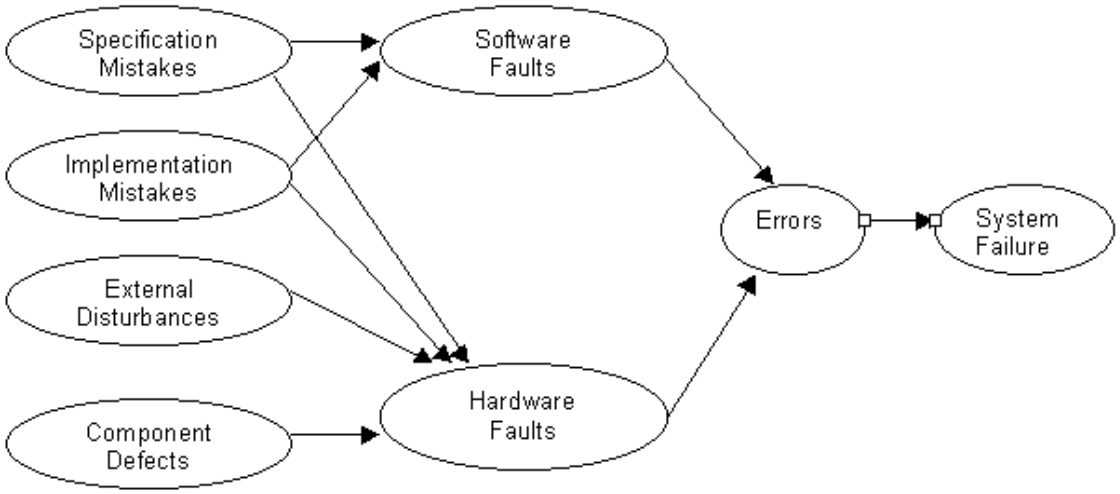
علت ديگر، خرابي عناصر است. ضعف‌هاي ساخت، خرابي تصادفي هر يك از اجزاء نمونه‌هايي از اين مورد هستند. اجزاء فيزيكي به راحتي ممكن است دچار خطا شوند و خرابي هر يك از اين عناصر مهمترين علت خطا است.

آخرين علت بروز خطا اغتشاشات خارجي محيط هستند. براي مثال تشعشعات و تداخل‌هاي الكترومغناطيسي يا اشتباهات اپراتوري و در نهايت عوامل محيطي مي‌توانند موجب خطا شوند. گاهي وجود عدم قطعيت در محيط و نپرداختن صحيح به اين مساله مي‌تواند موجب بروز خطا شود. اگر يك مدار الكترونيكي در يك كاربرد نظامي در سرماي بيش از اندازه قرار بگيرد، ممكن است نتايج غيرقابل قبول توليد نمايد. شكل 3، عللي كه بحث شد را به همراه ارتباط ميان آنها نشان مي‌دهد.

۱Specification

۲Implementation

۳External Factors



شكل 3- رابطه علت و معلولي ميان خرابي و خطا و علل بروز خطا در سيستم ها

۲-٦- ويژگي‌هاي خطا

در اين بخش براي مشخص‌تر شدن بحث در مورد خطا يك سري ويژگي‌هاي رفتاري و ذاتي را بيان مي‌كنيم:

طبيعت خطا، نوع خطا را مشخص مي‌كند. براي مثال آيا اين خطا سخت‌افزاري است يا نرم‌افزاري و يا آنالوگ است يا ديجيتال.

طول مدت[1] *خطا* ، طول مدت زماني كه خطا در سيستم فعال است را مشخص مي‌كند. اولين نوع خطا ، ماندگار[۲] است كه اگر عملي در قبال رفع آن انجام نگيرد، براي مدت طولاني در مدار باقي مي‌ماند. مورد بعدي خطاي گذرا[۳] است كه مي‌تواند در مدت كوتاهي از زمان ظاهر شده و دوباره از بين برود. مورد سوم خطاي تناوبي[٤] است كه در آن هر خرابي مي‌تواند ظاهر شده، از بين برود و دوباره به صورت متناوب اين رخداد تكرار شود.

مسأله *محدودة خطا*[٥] مشخص مي‌كند كه آيا خطا به يك واحد سخت‌افزاري / نرم‌افزاري محدود مي‌شود يا اينكه كل سيستم را فرا مي‌گيرد. مثلاً خرابي منبع تغذيه سيستم مي‌تواند به عنوان نمونه‌اي از خطاهاي فراگير بيان شود.

۱Duration

۲Permanent

۳Transient

٤Intermittent

٥Extent

۲-۷- فلسفه‌هاي طراحي براي فائق آمدن بر خطا

در حالت كلي سه روش عمده براي فائق آمدن بر خطا و نگه داشتن سيستم در كارايي نرمال خود وجود دارد كه عبارتند از :

- اجتناب از خطا[۱]
- پوشاندن خطا[۲]
- تحمل‌پذيري خطا[۳]

اجتناب از خطا، هر تكنيكي كه براي جلوگيري از رخداد خطا در سيستم انجام شود دربرمي‌گيرد، مثل: بررسي مجدد طراحي، آزمودن و ديگر روشهاي كنترل كيفي. اگر در بررسي مجدد طرح، يك مشكل در توصيف نيازهاي سيستم مشخص شده و برطرف گردد، مشكلات متعددي كه مي‌توانستند موجب بروز خطا شوند، قابل مرتفع نمودن خواهند شد؛ يا با آزمودن يك طرح، خطاهاي بسياري قابل تشخيص و سپس برطرف نمودن هستند.

پوشاندن خطا، هر فرآيندي كه موجب شود پس از بروز خطا ، سيستم لااقل با اشتباه مواجه نگردد را شامل مي‌شود.

تحمل پذيري خطا، توانايي سيستم است بر ادامه دادن به وظايف خود پس از اينكه در سيستم با خطا مواجه شديم. هدف نهايي تحمل پذيري خطا؛ جلوگيري از بروز خرابي در سيستم است.تحمل پذيري خطا با اعمال تكنيك‌هاي مختلفي قابل ايجاد در سيستم است مثلاً پوشاندن خطا يكي از روشهاي كنار آمدن با خطا در سيستم است و روش ديگر حذف عنصر خراب از كل سيستم است. تغيير ساختار فرآيند[۴]، حذف يك ماهيت خراب از سيستم و بازگرداندن مجدد سيستم به وضعيت كارآمد خود مي‌باشد. اگر تكنيك‌هاي تغيير ساختار اعمال شود؛ طراح بايد به سه فرآيند زير به خوبي پرداخته باشد:

۱-تشخيص خطا[۵]: فرآيند شناسايي خطا در سيستم كه معمولاً پيش از هر فرآيند ديگري بايد انجام بگيرد.

۱Fault Avoidance

۲Fault Masking

۳Fault Tolerance

٤Reconfiguration

٥Fault Detection

۲- تعيين موقعيت خطا[۱]: فرآيند مشخص نمودن اين مسأله است كه در كجا خطا رخ داده است و بنابراين رفع خطاي مناسب قابل پياده‌سازي است.
۳-محدود نمودن حوزة خطا[۲]: فرآيند ايزوله نمودن خطا است و جلوگيري از اينكه اثرات آن در كل سيستم انتشار يابد.
٤-برطرف نمودن خطا[۳]: فرآيند بازگرداندن وضعيت كاري و عملكردي مناسب به سيستم است پس از اينكه به علت خطا با مشكل مواجه شده است.
يك سيستم نوعي كه تحمل پذيري خطا را پشتيباني نمايد و از تكنيك سازمان دهي مجدد استفاده نمايد، مطابق زير عمل مي كند:
فرض كنيد كه يك خطا در سيستم رخ داده است. تكنيك‌هاي تشخيص خطا، مشخص مي‌نمايند كه خطا در سيستم وجود دارد و روال‌هاي تعيين موقعيت خطا ، منبع خطا را تعيين مي‌كنند. رفع خطا و سازمان‌دهي مجدد ، واحد خراب را از سيستم حذف مي‌كنند و آن را با يك مولفه سالم جايگزين مي‌نمايند و يا با كاهش قابل قبولي در كارايي، سيستم را به صورت فعال باقي نگاه مي‌دارند.

۲-۸- افزونگي به عنوان تكنيكي براي تحمل‌پذيري خطا

مهم‌ترين تكنيكي كه تا به حال براي تحمل‌پذيري خطا در سيستم‌ها به كار رفته است، استفاده از تكنيك افزونگي[٤] است. در اينجا ابتدا مفهوم افزونگي و مصاديق افزونگي را در سيستم شرح مي‌دهيم.
افزونگي، اضافه نمودن اطلاعات، منابع يا زماني بيش از آنچه كه براي عملكرد حالت عادي سيستم لازم است به اجزاء سيستم است. افزونگي در هريك از چهار مورد زير مي‌تواند مصداق پيدا نمايد:
افزونگي سخت افزاري: اضافه نمودن سخت افزار اضافه، غالباً به منظور تشخيص خطا يا تحمل پذيري خطا.
افزونگي اطلاعاتي: اضافه نمودن اطلاعات بيش از آنچه براي پياده‌سازي تابع مورد نظر لازم است، براي مثال

۱Fault Location
۲Fault Containment
۳Fault Recovery
٤Redundancy

كدهاي تصحيح خطا از نوعي از افزونگي اطلاعاتي بهره مي‌گيرند.

افزونگي نرم افزاري: اضافه نمودن نرم افزار، بيش از آنچه براي كاركردهاي سيستم لازم است.

افزونگي زماني: استفاده از زماني بيش از آنچه توابع سيستم لازم دارند كه اجرا شوند، به طوري كه عمليات تشخيص خطا و تحمل پذيري خطا قابل انجام باشد. در اين بخش هر مورد را به تفصيل شرح خواهيم داد:

۲-۸-۱- افزونگي سخت افزاري

تكرار نمودن فيزيكي سخت افزار، رايج ترين شكل افزونگي است كه در سيستم‌هاي امروزي هم استفاده مي شود.

همانطور كه مولفه‌هاي سخت افزاري روز به روز ارزانتر و قابل استفاده‌تر مي‌شوند، اعمال اين نوع افزونگي به سيستم آسانتر و عملي‌تر مي‌شود. در عمل سه نوع افزونگي سخت افزاري وجود دارد :فعال[۱] ، غيرفعال[۲] و تركيبي[۳] يا هايبريد.

در تكنيك‌هايي كه به نوع افزونگي غير فعال مجهز هستند، از مفهوم پوشاندن خطا استفاده مي‌شود كه از بروز خرابي در سيستم جلوگيري مي‌كنند. به عبارتي در اين روشها، بيش از آنكه به تشخيص خطا بها داده شود، به پوشاندن آن پرداخته مي‌شود. در روش فعال كه غالباً روش ديناميك هم ناميده مي‌شوند، تحمل پذيري خطا به وسيله تشخيص وجود خطا و انجام عملي در قبال آن انجام مي‌پذيرد، مثلاً حذف عامل خرابي و تغيير ساختار سيستم به نحوي كه مجدداً به مد عملياتي بازگردد.

افزونگي سخت افزاري فعال از تشخيص خطا و تعيين موقعيت آن و رفع خطا ، استفاده مي كند تا به تحمل پذيري خطا نائل آيد.

در روشهاي تركيبي كه ويژگي‌هاي جالب هر دو روش فوق را تركيب مي‌كنند و در واقع رايج ترين نوع افزونگي سخت افزاري هستند، بيشترين هزينه مصرف مي‌شود.

افزونگي سخت افزاري غير فعال به مكانيزم راي گيري[٤] بستگي دارد كه براي پوشاندن رخداد خطا استفاده مي‌شود.

۱Active

۲Passive

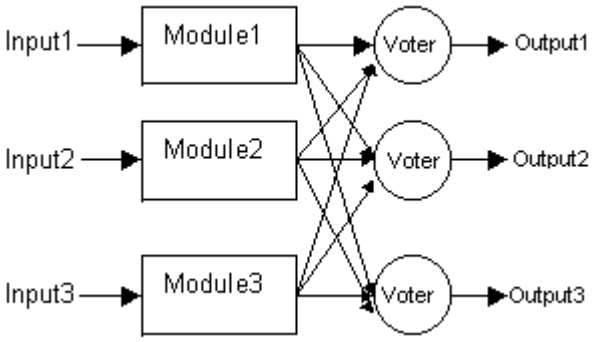
۳Hybrid

٤Voting

اكثر روشهاي غيرفعال از تكنيك رأي گيري براساس رأي اكثريت[1] استفاده مي‌كنند. مهم‌ترين روش براي اين نوع از افزونگي ،TMR[2] است. اساس اين روش اين است كه سخت‌افزار خود را به صورت افزونه سه تايي قرار دهيم و سپس براي تعيين خروجي نهايي از يك مكانيزم رأي گيري استفاده نمائيم. اگر يكي از ماجول ها با خطايي مواجه شود، دو ماجول ديگر كه بدون خطا هستند، خروجي نهايي را توليد خواهند نمود.

مشكل اساسي اين روش هنگامي است كه داور[3] با مشكل مواجه شود كه در آن صورت كل سيستم از كار خواهد افتاد. به عبارت ديگر، قابليت اطمينان اين نوع از T MR به هيچ وجه بهتر از قابليت اطمينان داور به تنهايي نيست.

هر مولفة منفردي كه خرابي آن موجب خرابي كل سيستم شود، نقطة منحصر به فرد خرابي[4] ناميده مي‌شود. يك روش براي بهبود اين مسأله ، استفاده از افزونگي در خود داور است كه در شكل 4 ديده مي شود.



شكل 4- TMR با افزونگي در خود Voter

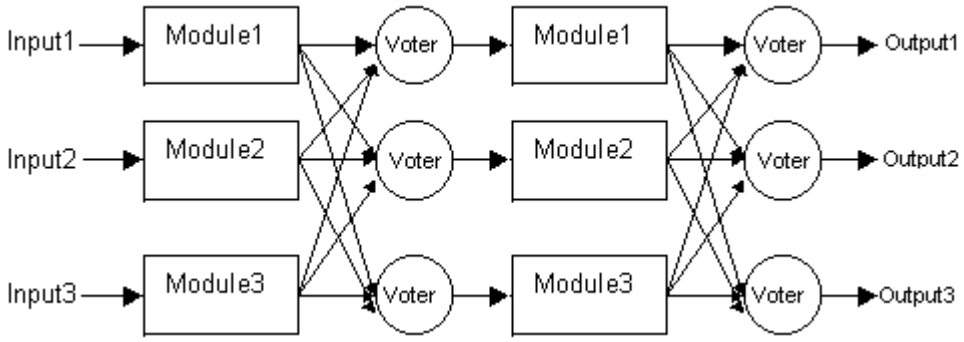
در صورتي كه پس از هر مرحله در سيستم از يك داور افزونه استفاده شود و خطاي هر مرحله پيش از انتقال به لايه بعد تصحيح شود، شكل مشابه شكل 5، حاصل خواهد شد كه TMR باچندگام ناميده مي‌شود.

۱Majority Voting

۲Triple Modular Redundancy

۳Voter

٤Single Point of failure



شكل 5- Multi-stage TMR (ايدة استفاده از Voter در هر مرحله).

در حالت كلي بجاي TMR مي توان از NMR[1] استفاده نمود. مهم ترين مصالحه در NMR درجه تحمل پذيري خطاي بدست آمده در مقايسه با ميزان سخت افزار لازم است.

در افزونگي سخت افزاري فعال، تلاش مي كنيم كه به كمك تشخيص خطا ، تعيين موقعيت خطا و رفع خطا به تحمل پذيري خطا دست بيابيم. اين گونه افزونگي در كاربردهايي بيشتر مورد توجه است كه مي‌توانيم سيستم را با تغيير ساختار در يك مدت زمان قابل قبول دوباره به حالت عملياتي مناسب بازگردانيم. از ساده ترين انواع اين نوع افزونگي مي‌توان به تكنيك سخت افزار مضاعف به همراه مقايسه[2] اشاره كرد.

در اين تكنيك دو ماجول مشابه به صورت موازي ، عمل يكساني را انجام مي‌دهند و سپس با انجام يك مقايسه در صورت بروز تفاوت، پيام خطايي صادر مي شود كه البته در اين نسخة ساده شده، معلوم نخواهد شد كه كدام عنصر با خطا مواجه شده است.

يكي ديگر از مشكلات اين روش آن است كه اگر هر دو مولفه به علت مواجهه با خطا، جواب يكساني توليد كنند به علت عدم وجود تفاوت، پيام خطايي صادر نخواهد شد در صورتي كه در اصل خطايي بروز كرده است. روش ديگري براي اعمال اين نوع افزونگي ، سخت افزار اضافه و آماده باش[3] است. در اين روش يك ماجول اصلي و فعال است و دومي به عنوان پشتيباني براي اولي قرار گرفته است.

۱N-modular Redundancy

۲Duplication With Comparison

۳Standby Sparing

وقتي كه بروز خطايي مشخص شد، ماجول اصلي با مورد پشتيبان جايگزين مي‌شود كه در اينجا فرآيند تغيير ساختار به شكل سوئيچ نمودن ميان اين دو انجام مي‌پذيرد. در صورتي كه بخواهيم اين زمان سوئيچ كوتاه باشد بايد از آماده‌باش‌گرم[1] استفاده نمائيم كه در آن، ماجول افزونه به صورت همزمان با ماجول اصلي ، وضعيت هاي سيستم را رديابي مي‌كند ودر هر لحظه آماده جايگزيني است. در حالي كه در نوع آماده‌باش‌سرد[2]، تا زماني كه نيازي نباشد ماجول افزونه خاموش است و بنابراين جايگزيني با آن صرف زمان بيشتري را مي‌طلبد.

روش بعدي زوج-ويك واحد اضافه[3] است كه در اين روش ويژگي هاي هر دو روش قبلي با هم آميخته است. در اين روش دو مولفه آماده باش استفاده مي‌شود و در عين حال دو ماجول هم به صورت موازي در زمان با هم فعالند و نتايج آنها مقايسه شده، خاصيت تشخيص خطا فراهم مي‌شود. سيگنال خطاي حاصل از مقايسه ، براي آغاز نمودن روال فرآيند تغيير ساختار كه ماجول‌هاي خراب را حذف مي‌كند و با واحد اضافه جايگزين مي‌نمايد بكار مي‌رود.

تكنيك آخر استفاده از Watchdog Timer است كه براي تشخيص خطا در سيستم‌هاي بسياري مورد استفاده قرار گرفته است. مفهوم اين تكنيك آن است كه از عدم رخداد يك اتفاق ، نتيجه مي‌گيريم كه در سيستم خطا رخ داده است. اين تايمر بايد به صورت پريوديك reset شود. هرخرابي در سيستم كه باعث شود اين عمل انجام نگيرد، موجب خواهد شد كه سيستم خاموش شده و به اين وسيله ديگر خرابي عمده‌اي رخ ندهد. فرض اساسي در اين تكنيك اين است كه سلامتي سيستم موجب مي‌شود كه اين تايمر به صورت متناوب reset شود.

در افزونگي سخت افزاري هايبريد از تركيب ويژگي‌هاي مفيد هر دو روش فوق استفاده مي‌شود و البته اين روش بسيار پرهزينه‌تر است و بنابراين در كاربردهايي به‌كار گرفته مي‌شود كه تحمل‌پذيري خطا بسيار ضروري مي‌باشد.

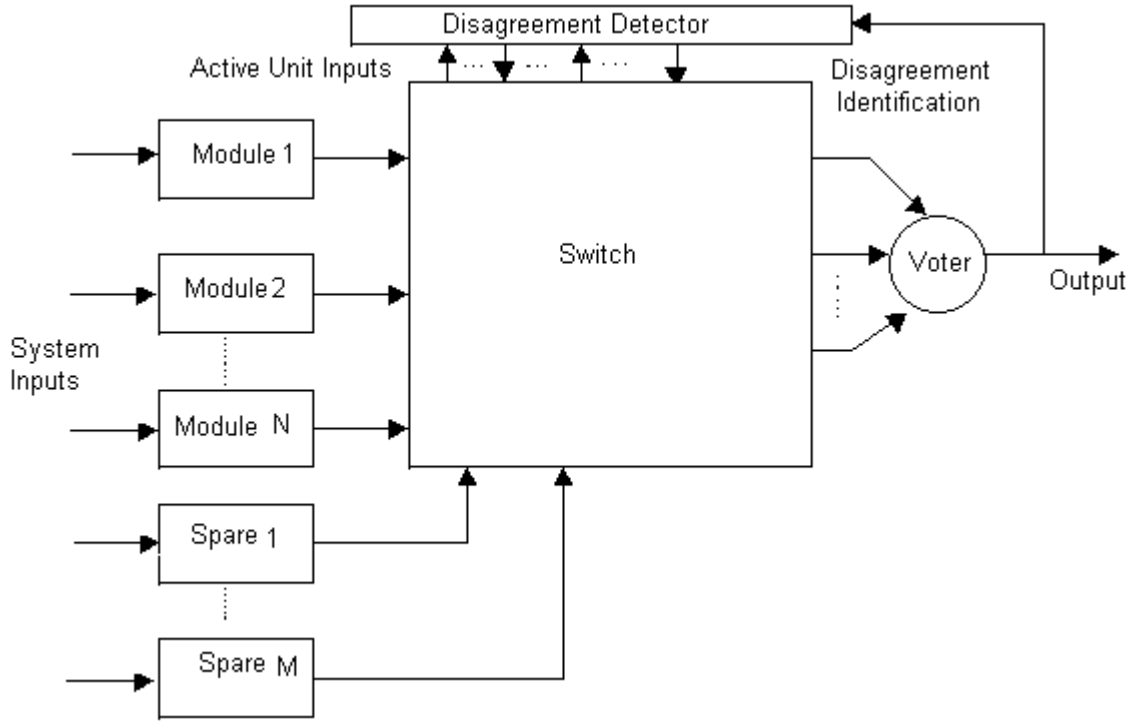
يكي از مهم‌ترين تكنيكهاي اين روش، N-modular Redundancy with spares است. اين روش هر دو ايدة NMR و Standby with spares را با هم در مي‌آميزد. شكل 6 بيانگر همين ايده است.

---

1 hot Standby Sparing

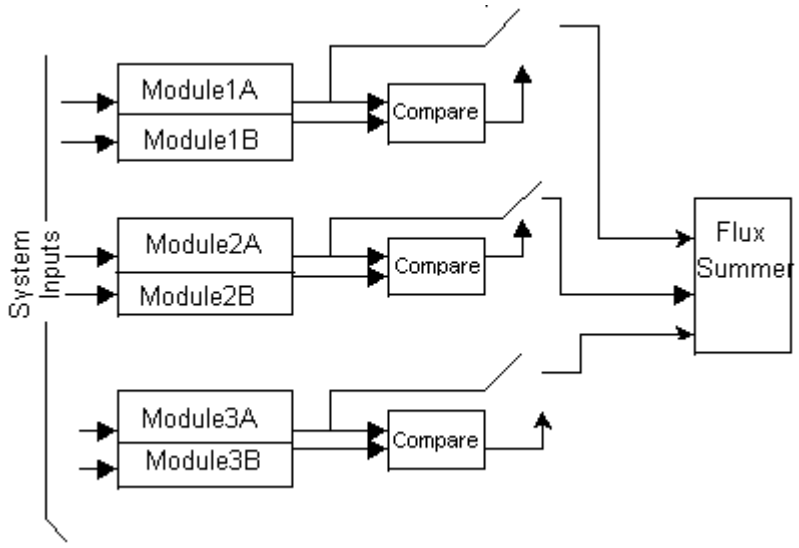
2 Cold Standby Sparing

3 Pair-and-a-spare



شكل 6- تركيب هر دو ايده NMR و Standby with Spare

روش بعدي معماري triple-duplex كه دو ايدة TMR و "سخت افزار مضاعف به همراه مقايسه" را باهم تركيب مي‌كند. استفاده از TMR اين امكان را فراهم مي‌كند كه خطا پوشانده شود ،در حالي كه استفاده از "سخت افزار مضاعف به همراه مقايسه" موجب مي‌شود كه خطاها تشخيص داده شود و ماجول خراب از فرآيند رايگيري حذف گردد. در شكل 7، اين ايده نمايش داده شده است.



شكل 7- معماري Triple-Duplex

۲-۸-۲- افزونگي اطلاعاتي

افزونگي اطلاعاتي به اضافه نمودن هر مقدار اطلاعات، بيشتر از داده هاي اصلي مسأله اتلاق مي‌شود به طوري كه تشخيص خطا و گاه تحمل‌پذيري خرابي انجام‌پذير باشد. مثال خوبي از اين نوع افزونگي، كدهاي تشخيص خطا و رفع خطاست كه به داده هاي اصلي اضافه مي‌شوند.

يك تكنيك اساسي كه در هر دو موضوع تشخيص خطا و نيز تصحيح خطا استفاده مي‌شود، كد Hamming و در اصل استفاده از فاصله Hamming است. از انواع كدهاي مورد استفاده در اين نوع افزونگي مي‌توان به موارد زير اشاره نمود:

- ✓ Parity code
- ✓ m-of-n codes
- ✓ Checksum
- ✓ Cyclic codes
- ✓ Residue codes
- ✓ Berger codes

۲-۸-۳- افزونگي زماني

مهم‌ترين مشكل استفاده از افزونگي‌هايي كه تاكنون شرح داده شدند نياز آنها به مقدار زيادي سخت افزار و بيت‌هاي اطلاعاتي در تكنيك‌هاي مختلف بود. اگر بخواهيم هر دو نوع منابع مورد نياز يعني داده و سخت افزار را كاهش دهيم، استفاده از زمان بيشتر بهايي است كه بايد بپردازيم.

در بسياري از كاربردها، زمان به اندازه سخت افزار ارزشمند نيست چرا كه براي سخت افزار بايدبه مسأله

هزينه، سايز، توان و وزن دستگاه‌ها توجه نمود. انتخاب نوع افزونگي لازم بستگي مستقيمي به نوع كاربرد دارد.

۲-۸-۳-۱- تشخيص خطاي گذرا

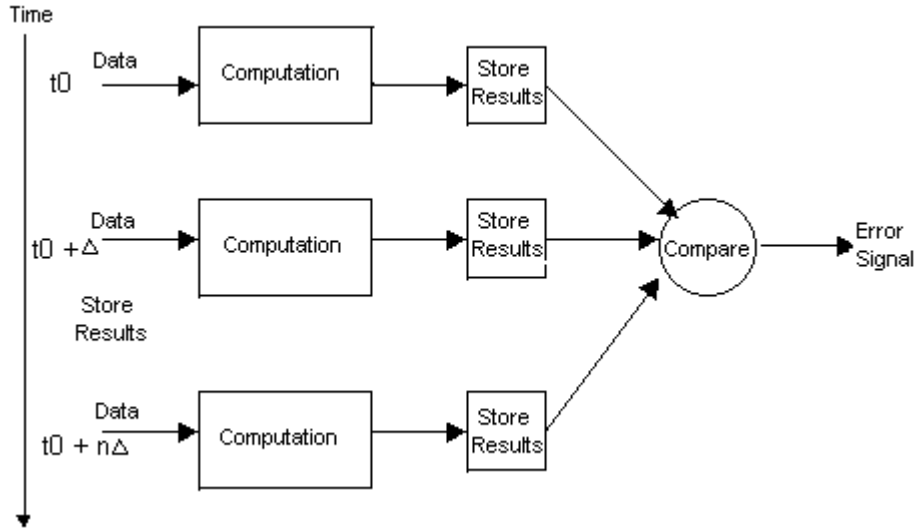
مفهوم اصلي در افزونگي زماني ، تكرار محاسبات در طول زمان است به نحوي كه وجود خطا را بتوان تشخيص داد ، يعني اگر يك كار را در طول زمان چند بار انجام بدهيم و سپس نتيجه را با هم مقايسه كنيم مي‌توانيم در صورت بروز تفاوت ، به وجود خطا پي ببريم و در اين صورت محاسبات را تكرار كنيم تا بفهميم كه آيا خطا هنوز وجود دارد يا از بين رفته است. شكل 8 استفاده از همين ايده را نشان مي‌دهد.

۲-۸-۳-۲- تشخيص خطاي ماندگار

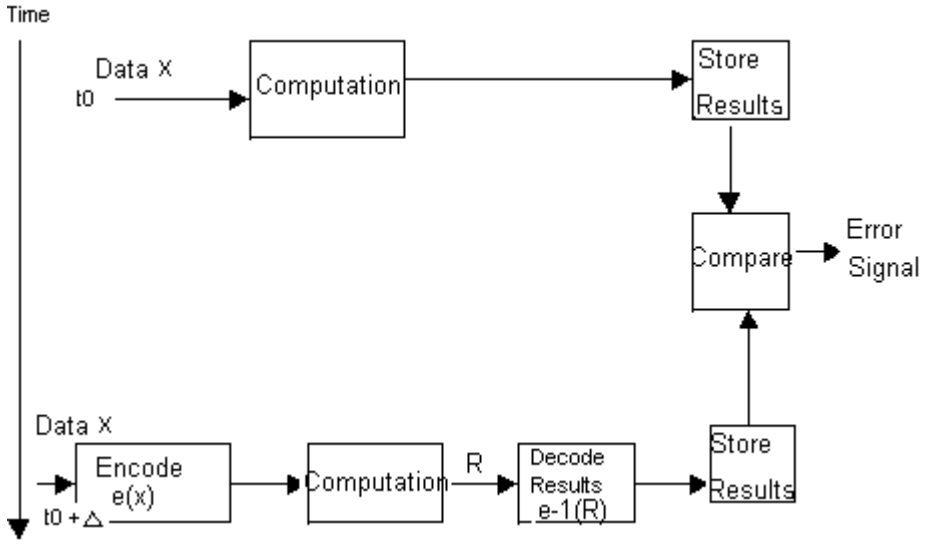
يكي از مهم‌ترين فواید استفاده از افزونگي زماني، توانايي اين روش در تشخيص خطاهاي ماندگار است . مفهوم اصلي مورد استفاده در اين روش در شكل 9 نشان داده شده است. در اولين مرحله محاسبه يا انتقال، اپرندها همانطور كه نمايش داده شده‌اند مورد استفاده قرار گرفته و نتايج در يك رجيستر ذخيره مي‌شوند. پيش از محاسبات بعدي يا انتقال، اپرندها رمز[۱] مي شوند (با استفاده از تابع رمز) . بعد از آن كه عمليات روي داده هاي رمزشده انجام گرفت ، نتايج رمزگشايي[۲] مي شوند و با نتيجه ، فاز اول مقايسه مي‌شود. در اصل تابع رمزكننده بايد طوري انتخاب شود كه عمل تشخيص خطا ممكن باشد. اين مفهوم را در شكل 9 مي‌توان مشاهده كرد.

۱Encode

۲Decode



شكل 8- افزونگي زماني و تكرار محاسبات و مقايسه نتايج



شكل 9- تشخيص خطاي ماندگار

۲-۸-٤- افزونگي نرم افزاري

در بسياري از سيستم‌هاي كامپيوتري، تكنيك‌هاي تحمل‌پذيري خطا مي‌تواند به صورت نرم افزاري پياده شود. سخت افزار افزونة لازم، براي پياده‌سازي توانمندي‌ها غالبا بايد مي‌نيمال باشد در حالي كه نرم افزار افزونه مي‌تواند با افزونگي بالاتري مورد استفاده قرار بگيرد.

اين افزونگي لزوماً به معناي تكرار كامل يك برنامه نيست بلكه مي‌تواند تنها به صورت تكرار چند خط برنامه يا تكرار يك روتين كوچك آن باشد . در اين بخش چند تكنيك در افزونگي نرم افزاري بيان مي‌شود.

۲-۸-٤-۱- آزمون سازگاري[۱]

يك آزمون سازگاري از دانش اوليه‌اي در مورد ويژگي‌هاي اطلاعاتي براي بررسي صحت اطلاعات استفاده مي‌كند. براي مثال در برخي كاربردها، از ابتدا مي‌دانيم كه بزرگي يك سيگنال از يك مقدار معلومي نبايد بيشتر باشد. چنين آزمونهايي با وجود اينكه در سخت افزار هم به راحتي قابل پياده سازي است، معمولاً به صورت نرم افزاري پياده مي‌شوند.

مورد ديگري كه براي آزمون سازگاري مي‌توان مثال زد و استفاده در كاربردهاي كنترلي دارد ، مقايسه مقدار اندازه گيري شده كارايي با چند مقدار پيش‌بيني شده است.

۲-۸-٤-۲- آزمون توانايي[۲]

آزمون‌هاي توانايي به اين منظور انجام مي‌گيرند كه مشخص نمايند،يك سيستم توانايي لازم را دارد يا خير. براي مثال ممكن است مايل باشيم بدانيم آيا كل حافظه، قابل دسترسي است و يا همة پردازنده‌هاي سيستم به درستي كار مي‌كنند يا خير؟

مدل‌هاي مختلفي براي آزمون توانايي وجود دارد. اولين آنها آزمون سادة حافظه است. مثلاً يك پردازنده مي‌تواند يك الگوي مشخص را در مكان معلومي از حافظه بنويسد و آن موقعيت را بخواند تا مطمئن شود كه آيا آن داده به درستي ذخيره و بازيابي شده است يا خير.

شكل ديگري براي آزمون توانايي ، به اين منظور است كه مثلاً بررسي كنيم كه آيا كلية پردازنده‌هاي يك سيستم چند پردازنده‌اي، قادرند به درستي با هم ارتباط برقرار نمايند يا خير. براي اينكار مي‌توان به صورت متناوب ، اطلاعات مشخصي را از يك پردازنده به ديگري منتقل نمود. براي مثال هر پردازنده مي‌تواند يك بيت خاص از يك حافظه اشتراكي را Set نمايد تا قالبيت ارتباط او با حافظه و در نتيجه با بقيه مشخص شود.

۲-۸-٤-۳- برنامه نويسي چندنسخه‌اي[۳]

برنامه نويسي چندنسخه‌اي به اين منظور توسعه يافته است كه به تشخيص مشكلات در برخي طراحي‌ها كمك نمايد و مفهوم اصلي آن، اين است كه يك برنامه را N دفعه توسط برنامه نويسان مختلف كد كرده باشيم و نتايج اين N

۱Consistency check

۲Capability check

۳N-Version Programming

برنامه را با هم مقايسه كنيم، به اين اميد كه حتماً N برنامه نويس اشتباهات مشابهي انجام نداده اند و بنابراين زماني كه يك خطا رخ مي‌دهد، يا در همه N ماجول رخ نمي‌دهد يا به صورت‌هاي متفاوتي رخ مي‌دهد و بنابراين نتايج مشابهي حاصل نخواهد شد . دو مشكل اصلي براي اين روش متصور است:

يكي اينكه لزوماً N دسته برنامه‌نويس اشتباهات كاملاً متفاوتي مرتكب نخواهند شد و بنابراين لزوماً اينطور نيست كه تضمين كرده باشيم كه دو نسخه كاملاً متفاوت از يك برنامه ، اشتباهات مشابهي ندارند و دومين مشكل اينكه اين N نسخه همگي از روي يك توصيف نوشته شده اند و بنابراين لزوماً تشخيص خطاهاي مختلفي را امكان پذير نمي‌سازند. به اين منظور مي‌توان از برنامه نويسان خواست كه هر يك قواعد طراحي مشخص و متفاوتي را اعمال كنند تا پياده سازي آنها تا حد قابل قبولي متفاوت باشد [۱] [۲] [۳] [۴] .

۳- مرور كارهاي انجام شده در زمينة تحمل‌پذيري خطا در سيستم‌هاي چندعامله

در فصل قبل مفاهيم تحمل‌پذيري‌خطا به صورت كلي بيان شدند و تكنيك‌هاي عملي و رايج تحمل‌پذيري‌خطا در سيستم‌ها شرح داده شدند. اكنون در اين فصل با توجه به ويژگي‌هاي خاص يك سيستم چندعامله به عنوان نمونة مهمي از سيستم‌هاي گسترده، تحقيق‌هايي كه تا به‌حال در مورد اين سيستم‌ها با نظر به مفاهيم مرتبط با تحمل‌پذيري خطا انجام شده است را مورد بررسي قرار مي‌دهيم. البته لازم به توضيح است كه در هيچ يك از اين تحقيق‌ها به صورت دقيق به مسالة تحمل‌پذيري خطا پرداخته نشده است، بلكه در اغلب موارد تشخيص خطا مد نظر قرار گرفته است و موارد معدودي كه مشخصا با هدف تحمل‌پذيري خطا انجام گرفته‌اند، روشهايي مبتني بر افزونگي را به عنوان راه حل پيشنهاد كرده‌اند. بنابراين براي مشخص‌تر نمودن تفاوت‌هاي كارهاي مشابه و آنچه در اين تز مد نظر است، در انتهاي اين فصل و در بخش نتيجه‌گيري مقايسه‌اي اجمالي انجام شده است و تمايز روشهاي پيشنهادي با روشهاي مطرح شده در اين فصل، بيان شده است.

۳-۱- مقدمه

يك سيستم چندعامله مجموعه‌اي است متشكل از چند عامل كه يك وظيفه مشخص را با همكاري انجام مي‌دهند. فايده داشتن يك سيستم چند عامله، توانايي اكتساب بازنمايي‌هاي دانش گوناگون و تسهيل همكاري ميان اين بازنمايي‌هاي مختلف است. ايده اصلي، داشتن اجتماعي از عاملهاست كه هر يك وظيفه مشخصي را انجام مي‌دهند و در تعامل با بقيه عاملها به يك هدف نهايي نائل مي شوند. به اين منظور لازم است كه دانش را تكرار[1] نمود يا به بيان صحيح‌تر به هر عامل دانشي از خودش و از كل اجتماع داد.

با داشتن چنين سيستمي، عاملها محيط را حس مي‌كنند (يا به طور مستقيم و يا با اكتساب دانشي از بقيه عاملها) و با تشكيل گروههاي همكاري و اعمال محرك به محيط ، با محيط ارتباط برقرار مي‌نمايند [۵] [۱۲] .

بنابراين سيستم‌هاي چند عامله هم مانند بقيه سيستم‌هاي توزيع شده، بسيار با خطا مواجه مي‌شوند.

---

۱ Replicate

عامل‌ها و منابع ممكن است به علل مختلفي همچون خرابي بستر ارتباطي ، قطع ارتباط، خرابي فرآيند و بسياري علل سخت افزاري يا نرم افزاري ديگر با مشكل مواجه شده و براي مدتي دسترس‌ناپذير بشوند. اغلب كارهاي انجام شده در زمينة تحمل‌پذيري‌خطا در سيستم‌هاي چندعامله براي تشخيص و تصحيح ، از تكنيك‌هاي رايجي كه در همه سيستم‌هاي گسترده استفاده مي‌شود، استفاده مي كنند [۸] .

در مورد ايجاد تحمل پذيري خطا در يك سيستم چندعامله، دو ديدگاه متفاوت وجود دارد:

اولين نظريه مي گويد كه چون سيستم هاي چندعامله ساختار، ماجولار دارند، ذاتاً تحمل پذير خطا هستند و يك خطا كه در يك ماجول بروز نمايد مي تواند از كل سيستم ايزوله شود و در سيستم انتشار نيابد. نظريه دوم بر اين باور است كه سيستم هاي چندعامله ذاتاً امن نيستند و چون كنترل در آنها توزيع شده است و غيرقطعي هستند، نمي‌توان يك رفتار را به ويژه در شرايط خطا تضمين نمود [۷] .

۳-۲- تحقيقات انجام‌شده در زمينة تشخيص خطا و استفاده از افزونگي براي ايجاد تحمل‌پذيري خطا

در بسياري از تحقيقات انجام شده در حوزه‌هاي MAS[۱] و DAI[۲] عموماً فرض مي‌شود كه عامل‌ها بدون خرابي هستند و تئوري‌ها و معماري‌هاي ارائه‌شده از بحث در مورد عامل‌هاي دچار اشكال اجتناب مي‌كنند. در ادامه چند نمونه از تحقيقاتي كه تحمل‌پذيري‌خرابي در سيستم‌هاي چندعامله نرم‌افزاري را به صورت محدود و در شرايط خاص فراهم آورده‌اند ذكر شده و سپس چند سيستم عامل‌گرا با توضيحات بيشتر معرفي مي‌شوند.

Deen محيطي توزيع شده براي سيستم‌هاي توزيع‌شده همكار پيشنهاد داده است [۱۶] . هر يك از اين سيستم‌ها به عنوان يك عامل خودمختار در نظر گرفته مي‌شود و مجموعه عامل‌ها براي حل مسائل كاربردي واقعي كه طبيعتي گسترده دارند با هم همكاري دارند. در اين محيط تحمل‌پذيري خرابي با جايگزيني عامل‌هاي خراب انجام مي‌شود و پس از جايگزيني عامل در برنامة گروه تجديدنظر صورت مي‌گيرد.

۱-Multi agent system

۲-Distributed artificial intelligence

Vogler و همكاران در [۱۷] [۱۸] يك معماري براي سيستمي از عامل‌هاي متحرك پيشنهاد داده‌اند. اين معماري با هدف فراهم آوردن امنيت در فرآيندهايي كه توسط عامل‌ها در تجارت الكترونيك و مبادلات پولي انجام مي‌شود ارائه شده است. براي رسيدن به اين هدف از مكانيزم‌هاي مختلف كدكردن اطلاعات استفاده شده است.

Strasser دو روش براي افزايش تحمل‌پذيري خرابي در عامل‌ها ارائه داده است [۱۹] . استفاده از مفهوم خطسير[۱] اين امكان را مي‌دهد كه حركت عامل به صورت انعطاف‌پذير بيان شود و به عامل‌ها امكان مي‌دهد كه عزيمت به گره‌هايي كه در دسترس نيستند را به تعويق انداخته يا گره‌هايي ديگر را به جاي آنها انتخاب كنند. روش ديگر استفاده از يك پروتكل تحمل‌پذيرخرابي است تا اطمينان حاصل شود كه هريك از عامل‌ها تنها يك بار اجرا مي‌شوند. عامل‌ها در اين پروتكل در مراحل[۲] مختلف اجرا مي‌شوند. هر منزل شامل تعدادي گره است و در هر منزل تنها يك گره مي‌تواند عامل را اجرا كند و ساير‌ين بر اين عمليات نظارت دارند. نظارت ساير عامل‌ها سبب آشكارسازي خرابي‌ها مي‌شود.

Stoller در [۲۰] روشي خودكار براي تحليل تحمل‌پذيري خرابي سيستم‌هاي توزيع‌شده ارائه داده است. روش بر پايه مدل Stream در محاسبات است كه از مكانيزم‌هاي تخمين هم بهره مي‌گيرد. كاربردي نوعي كه از اين روش عنوان شده است تحمل‌پذيري خرابي در عامل‌هاي متحرك مي‌باشد.

[۲۱] يك پروتكل تحمل‌پذيرخرابي معرفي نموده است كه در برابر خرابي‌هاي طولاني مدت گره‌هاي شبكه مقاوم است. اگر يك گره كه اجراي عامل در آن انجام مي‌شود دچار خرابي شود مي‌توان عمليات را ترميم نموده و در يك گره ديگر ادامه داد. در اين استراتژي اطلاعات لازم براي ترميم عمليات به صورت توزيع‌شده ذخيره مي‌شوند، وبنابراين هريك از گره‌هاي شبكه مي‌توانند بخشي از عمليات را ترميم كنند.

در سيستم ARCHON كه در [۲۲] معرفي شده است، مسأله هماهنگي سراسري بين عامل‌ها بررسي شده ولي بحثي ازعامل‌هاي معيوب به ميان نيامده است. درسيستم PRS كه درتعدادي كاربردهاي تجاري استفاده مي‌شود هم هيچ روشي براي مديريت خرابي وجود ندارد. خرابي فقط از ديدگاه سيستم ارتباطي وسيستم‌عامل مورد توجه قرارگرفته است و براي اشكالات ممكن

۱-Itinerary

۲-Stages

درفازهاي توصيف، طراحي وپيادەسازي سيستم ابزارهاي آزمون وعيب‌يابي خاصي مورداستفاده قرارمي‌گيرد.

در [۲۳] قابليت سازمان‌دهي مجدد در مدل يادگيري سازماني بررسي شده است. اين عمل با هدف حفظ كارآيي مجموعه عامل‌ها در مواقع بروز خرابي صورت مي‌گيرد. هنگامي كه در گروه رباتها تغييري بوجود آيد (بدين صورت كه تعدادي ازآنها معيوب شوند ويا از كار بيفتند) عامل‌ها بايد با تغيير سازمان سعي كنندكه وظيفة گروه را به انجام برسانند. سيستم رده بندي مبتني بر يادگيري سازماني مدلي جديد براي سيستم هاي چند عامله مي‌باشد. اين سيستم مركب از يك بخش توليد كنندة دستور بعلاوة مكانيزمي براي يادگيري تقويتي است. عليرغم اينكه اين روش بر پايه روشهاي يادگيري تكاملي بنا شده است ولي از روشهاي رايج يادگيري تكاملي از اين جهت متفاوت است كه جمعيتها را با توابع سراسري مانند تابع Fitness ارزيابي نمي‌كند. در اين روش رباتها رفتار خود را با استفاده از توابع محلي ارزيابي مي‌كنند.

در اين مدل رباتها رفتار مطلوب را براي انجام وظايف محوله به تنهايي و بر اساس مدل خود انتخاب مي‌كنند. در صورتي كه تعدادي از رباتها ازگروه حذف شوند يا توان ادامة فعاليت را نداشته باشند، اتمام عمليات براي ساير‌ين دشوار مي‌شود. دراين حالت رباتها بايد با استفاده از مدل يادگيري سازماني سعي كنند رفتار مناسب را انتخاب كرده و وظايف گروه را با همكاري ديگران به اتمام برسانند.

طرحي كه پياده سازي شده به اين صورت است كه پس از بوجود آمدن تغيير در گروه رباتها و ناتواني گروه در انجام وظيفه، رباتها در چند مرحله اقدام به يادگيري كرده و سعي مي‌كنند با تغيير سازمان گروه و نحوة تخصيص وظايف، مأموريت را انجام دهند. ممكن است گروه در چند گام اول موفق نشوند ولي با سعي بيشتر و تكرار يادگيري موفق به انجام مأموريت خواهدشد. اين روش توسط Kasahara پيادەسازي شده و نتايج آن گزارش شده است.

روش فوق براي سيستم هاي زمان‌حقيقي[1] روش مناسبي نيست. چون در اين سيستم ها زمان اهميت زيادي دارد و علاوه برآن با توجه به اهميت ميدان عملياتي كه خيلي از فرايندها برگشت‌پذير نيستند جايي براي يادگيري وجود ندارد. در

[1] RealTime Systems

جوامع انساني هم وضع به همين صورت است و سعي مي‌شود كه براي كارهاي خطير از افراد باتجربه استفاده شود.

۳-۳- ايجاد تحمل پذيري خطا به كمك تغيير نقش هر عامل [۵] [۱۰] [۱۱]

با در نظر گرفتن نقشي[1] براي هر عامل در يك سيستم چندعامله، ايدة داشتن سيستم هاي چندعامله به صورت خود ساختار دهنده[2] امكان پذير مي‌شود. اين رفتار پيچيده موجب مي‌شود كه بتوان سيستم هاي كنترلي در غالب سيستم هاي چندعامله با ويژگي تحمل پذيري خطا داشت.

تفاوت استفاده از سيستم هاي چندعامله براي تشخيص خطا و مونيتورينگ و اينكه با استفاده از سيستم هاي چندعامله، تحمل‌پذيري خطا به وجود آورده شود، بسيار ظريف و در عين حال بسيار مهم است: در مورد اول، عامل‌هاي استاتيك براي وظيفه هاي خاصي تعريف مي شوند مثلاً عاملهايي براي كارهايي از قبيل: جمع آوري هشدار. در حالي كه در مورد دوم ، خود عامل‌ها تشخيص مي‌دهند كه شامل خطا هستند و آنرا برطرف مي كنند. در حوزة كاربرد سيستم هاي چندعامله، تحقيقاتي انجام شده كه از عاملها براي افزونگي ماجولار استفاده نموده ولي در زمينة تغييرات ساختاري به علت بروز خطا كار جدي انجام نشده است.

اساسي‌ترين خاصيت تحمل پذيري خطا كه به وسيله يك سيستم چندعامله ارائه مي‌شود، از لحاظ مفهومي شبيه است به آنچه در بخش قبل با عنوان افزونگي آماده‌باش[3] شرح داده شد.

در يك سيستم چندعامله، ايده متفاوت براي تحمل پذيري خطا اين است كه وقتي عاملها، يك خطا را در يك سيستم چندعامله مشخص نمودند، سيستم را طوري تغيير ساختار بدهند كه يا خطا را برطرف نمايند يا به نحوي با آن كنار بيايند. منظور از تغيير ساختار[4]،تغيير عامل‌هايي است كه با آنها در همكاري هستند و يا تغيير در تعداد عامل‌هاي سيستم گسترده مي‌باشد.

۱Role

۲Self-Structuring

۳Standby Sparing

٤Restructure

۳-۳-۱- ويژگي‌هاي تحمل‌پذيري خطا

همانطور كه شرح داده شد، سيستم چندعامله‌اي با اين طراحي، قادر به تغيير ساختار پوياي خود مي‌باشد؛ و همين خاصيت است كه ويژگي‌هاي تحمل‌پذيري خطا را پديد مي‌آورد. در اين بخش اين ويژگي‌ها را شرح مي‌دهيم.

۳-۳-۱-۱- خودمختاري

خودمختاري به وسيلة برنامه‌ريزي عامل به طوري كه وقتي ارتباطش با ديگر عاملها قطع شد، به طور منطقي تصميم بگيرد، تحقق مي‌يابد. مثلاً هنگامي كه ارتباط از دست برود، نقش هر عامل تغيير كند و احياناً ارتباط با عامل ديگري را به نحو ديگري مورد توجه قرار بدهد.

۳-۳-۱-۲- ساختار پويا

با استفاده از نقش هر عامل ، اين تغيير ساختار پويا امكان‌پذير مي‌شود. اگر به هر دليل يك عامل تشخيص بدهد كه قادر نيست نقش فعلي خود را به انجام برساند (به علت از دست دادن ارتباط يا مشخص شدن خرابي فعال) عامل مي‌تواند:

- به دنبال روش جايگزين باشد كه نقش خود را با برقراري ارتباط با ديگر عاملها به انجام برساند.
- درخواست نمايد كه عاملي كه باعث به تعويق انداختن تكميل نقش او شده است، از بين برود و يا از گروه كاري حذف شود.
- نقش خود را تغيير بدهد، طوري كه يا با وجود خطا كار كند يا با آن كنار بيايد.

در [٥] يك كنترلر PID چند عامله طراحي شده است كه نشان مي‌دهد چگونه يك تغيير در نقش عامل مي‌تواند يك سيستم چند عامله را تطبيقي نمايد.

۳-٤- تحمل‌پذيري خطا به كمك سرويس‌هاي رفع خطا[1]

در[٦] يك سري سرويس‌هاي مستقل از دامنه[2] براي رفع خطاهايي مانند حالات استثنايي كه ممكن است در محيط رخ بدهد، با اعمال نمودن تغييراتي به پروتكل هماهنگسازي معروف تيم‌هاي چندعامله يعني Contract Net معرفي شده است. نشان داده‌شده است كه اين سرويس‌ها رفتار بسيار مؤثرتري

۱Fault-Handling

۲Domain independent

در زمينه تحمل‌پذيري خطا پديد مي‌آورند و نسبت به تكنيك‌هاي موجود براي تحمل‌پذيري خطا از كارايي بهتري برخوردارند.

به عللي كه در اينجا بيان مي‌شود، ممكن است نتوان با اعتماد به عملكرد مناسب عاملها و محيطي كه در آن عمل مي‌كنند، محيط را ايده‌آل فرض نمود و بايد به يك سري حالت خاص كه در سيستم ممكن است رخ بدهند و اگر به خوبي رفع نشوند، كارايي نامناسب و حتي خرابي كامل سيستم را موجب مي‌شوند، توجه نمود:

۱- زيرساختار نامطمئن[۱]: در سيستم‌هاي بزرگ توزيع شده‌اي مانند اينترنت، خرابي غيرقابل پيش‌بيني مانند خرابي هر گره و يا اتصالات ممكن است موجب مرگ ناگهاني عامل‌ها شود يا باعث شود كه پيام‌ها، دير ارسال شوند و يا گم شوند.

۲- عامل‌هاي غيرهمخوان[۲]: در سيستم‌هاي باز، عامل‌ها جداگانه توسعه مي‌يابند و به طور آزاد، رفت و آمد مي‌كنند و بنابراين هميشه نمي‌توان به آنها اعتماد نمود كه به قوانين عمل كنند.

۳- خرابي‌هاي اضطراري

در [٦] يك سري سرويس‌ها به منظور رفع خطا معرفي مي‌شود كه به پروتكل CNET اعمال شده است.

CNET با سه دستة مهم از حالات خاص مواجه است:

مرگ عامل[۳]: اگر يك عامل CNET بميرد، نتايج ناگهاني بسيار متفاوتي حاصل مي‌شود. اگر اين عامل به عنوان كارگزار در حال فعاليت بوده باشد، حتماً مشتري آن، نتايج مورد انتظار را دريافت نخواهد كرد. علاوه بر اين اگر اين عامل خود چند كارگزار را براي چند زيروظيفه به كار گماشته باشد، اين زيروظيفه و همه زير وظايف آن يتيم خواهند شد و ديگر هيچ دليلي براي تكميل آنها باقي نمي‌ماند، در حالي كه از سوي ديگر در حال مصرف كردن منابع و حافظه سيستم هستند.

عامل فريبكار[٤]: يك عامل CNET كه داراي مشكل نرم‌افزاري باشد مي‌تواند به دروغ يا اشتباه تبليغاتي در مورد توانايي‌هايش بكند و اين باعث شود كه روند كاري

---

۱ Unreliable Infrastructure

۲ Non-Compliant

۳ Agent Death

٤ Fraudulent Contractor

سيستم مختل شود. مثلاً اگر يك عامل به عنوان كارگزار ادعا كند كه هر وظيفه را مي‌تواند با سرعت انجام بدهد و سپس به عنوان برنده انتخاب شود و بعد نتايج نامناسب و اشتباهي برگرداند، نتيجه اين است كه تمام يا اكثر وظايف سيستم به يك عامل ناكارا ارجاع بشود.

اتلاف منابع[1]: سيستم هايي كه از CNET استفاده مي‌كنند مي‌توانند به وظايف يك سري اولويت نسبت بدهند و بنابراين وقتي يك كارگزار با چند RFB[2] مواجه مي‌شود آن RFB كه بالاترين اولويت را دارد، انتخاب نمايد. يكي از معايب اين كار اين است كه ممكن است اتلاف منابع رخ بدهد. در واقع در حالتي كه يك وظيفة كم اولويت ولي طولاني داريم، كل منابع سيستم كه براي وظيفة پراولويت‌تري كه بعداً مي‌رسند لازم خواهند بود، را درگير مي‌كند. اين مسأله موجب كاهش چشمگير كارايي در سيستم هاي پيچيده مي‌شود. مكانيزم استاندارد براي رفع حالات خاص در CNET مانند اكثر پروتكل‌هاي توزيع شده Time out/Retry است. يعني اگر هيچ نتيجه اي تا موعد مقرر مربوط به كارگزار باز نگردد، يك كارفرما، فرآيند كارگزار را دوباره با ارسال RFB جديد آغاز مي كند. اين كار مشكل ناشي از مرگ عامل را حل مي كند ولي بسيار ناكاراست چرا كه وظيفه هاي يتيم شده را حذف نمي كند و باعث رخداد خاصيت جمع‌شدن[3] Retry / timeout ها مي شود. راه‌حلهاي برطرف نمودن كاستي‌هاي CNET در [۶] از طريق روش پيشنهادي Citizen حل شده است. اين نامگذاري به خاطر شباهتي است كه ميان اين روش رفع خطا و روشي كه خطا در جوامع انساني برطرف مي شود، وجود دارد.

۳-۵- تحمل پذيري خطا به كمك عاملهاي محافظ[4]

در[۷] عامل‌هاي محافظ به عنوان روشي براي ايجاد تحمل‌پذيري خطا در سيستم هاي چند عامله پيشنهاد شده است. عامل‌هاي محافظ ، به منظور حفظ يك سري ويژگي‌ها در سيستم و نيز جلوگيري از ورود سيستم به وضعيت‌هاي ناخواسته در سيستم قرار مي‌گيرند. عامل‌هاي محافظ يك ساختار كنترلي براي سيستم هاي چندعامله پديد مي‌آورند و ارتباطات را مانيتور مي‌كنند و مدلي از ديگر عاملها مي‌سازند و با توجه به

۱Resource Poaching

[2] Request for Bid

۳Cascad

٤Sentinel Agent

راهكارهاي داده شده با آنها تعامل مي‌كنند. از آنجا كه محافظ‌ها خود عامل هستند، با عامل‌هاي ديگر به وسيله زبان‌هاي ارتباطي ميان عامل‌ها ارتباط برقرار مي‌كنند. علت استفاده از اين عامل‌ها آن است كه آزادي عاملها را در هر كجا كه لازم بدانيم به منظور حفظ جامعيت سيستم محدود نمائيم ، ولي اين كنترل آزادي بايد ساده و قابل تغيير باشد.

محافظ، يك عامل است و هدف آن محافظت از برخي توانمندي‌هاي سيستم در برابر بعضي وضعيت‌ها در اجتماع عامل‌هاست. محافظ، خودش مستقيماً در حل مسأله دخالت نمي‌كند، اما اگر لازم باشد مي‌تواند در مسائلي از قبيل انتخاب راه ديگري براي حل مسأله توسط عامل‌ها، خارج كردن عامل خراب از سيستم، تغيير پارامترها براي عامل‌ها و اعلام گزارش به اپراتور دخالت كند. بنابراين با مشاهدة ارتباط ميان عامل‌ها و با تعامل (پرسيدن) ، مدل‌هايي از ديگر عامل‌ها مي‌سازد. در ضمن مي‌تواند با استفاده از تايمر، عامل‌هاي خراب يا خرابي بستر ارتباطي را مشخص نمايد.

با داشتن مجموعه‌اي از عامل‌ها كه در تحقق يك تابع همكاري مي‌كنند، يك محافظ به عنوان يك پاسدار از عمل كردن آن تابع محافظت مي‌كند. محافظ مدل‌هايي از عاملها نگهداري مي‌كند. بعضي از موارد در اين مدل مستقيماً از مدل جهان عامل كپي مي‌شوند و خود جزئي از مدل جهان مربوط به محافظ مي‌شوند، اين موارد را Chekpoint مي‌ناميم. استفاده از Chekpoint به محافظ اجازه مي‌دهد كه براساس آنها در مورد وضعيت عامل‌ها قضاوت نمايند. اين يك روش اوليه براي شناخت عامل خراب است كه براي شناخت تضاد ميان عاملها كه خود يك خرابي در سيستم است، نيز كاربرد دارد.

۳-۶- تحمل پذيري خطا به كمك تيمي از عامل‌هاي واسط[1]

در [۸] [۹] استفاده از كارتيمي براي ساختن معماري مقاومي كه بتواند بر خرابي‌هايي كه براي واسطها به وجود مي‌آيد، فائق گردد پيشنهاد شده است. با استفاده از عملكرد تيمي همواره به صورت تضمين‌شده‌اي تعدادي واسط فعال

[1] Broker

در سيستم چند عامله داريم كه عاملهاي ديگر با آنها در ارتباطند.

ايده اصلي آن است كه هر عامل بايد با يك واسط كه خود يك عامل مياني است، رجيستر شده باشد. اگر به هر دليلي آن واسط غيرقابل دسترسي بشود، عامل به دنبال واسطي مي‌گردد و اين كار را به كمك مخابره كردن پيام خود انجام مي‌دهد. سيستم‌هاي چندعامله براي انجام وظيفه‌هاي خاصي مانند پذيرش درخواستها، يافتن عامل با قابليت مورد نظر، مسيريابي درخواستها اشتراك اطلاعات و مديريت سيستم و رجيستر نمودن قابليت عاملها به عامل مياني[1] كه واسط هم ناميده مي‌شود، نياز دارند. هر چه تعداد عاملها بيشتر مي‌شود و به عبارتي سيستم چندعامله پيچيده‌تر مي‌شود، تعداد واسطهاي لازم در سيستم افزايش مي‌يابد. در يك سيستم چند عامله بزرگ با واسطهاي چندگانه، واسطها را مي‌توان به عنوان پشتيبان‌هايي براي يكديگر بكار برد و بنابراين تحمل پذيري خطاي بهتري بدست آورد.

هنگامي كه يك محافظ از هم تيمي‌هاي خود قطع ارتباط نمود، همه محافظهاي آن تيم تلاش مي‌كنند كه با عاملهايي كه با آن رجيستر شده‌بودند، ارتباط برقرار نمايند. وقتي كه يك محافظ، به اين شكل به طور موفقيت‌آميز با يك عامل ارتباط برقرار نمود، به هم تيمي‌هايش اطلاع مي‌دهد و بقيه محافظها تلاش‌هايشان را براي ارتباط با آن متوقف مي‌كنند.

۳-۷- تحمل پذيري خطا در تيمي از رباتهاي همكار [۱۵]

رفتارتحمل‌پذير خرابي‌در ربات متحرك به معناي آشكارسازي و تشخيص خرابي‌ها به صورت خودكار و توانايي ادامة فعاليت پس از بروز خرابي مي‌باشد. [۱۵] به دو مولفة اول (آشكارسازي وتشخيص خرابي) مي‌پردازد و هدف آن توسعة روشي است كه رباتهاي متحرك را در آشكارسازي، تشخيص و بر طرف كردن خرابي‌ها (كه نتيجة اشكال سنسورها، محركها و اجزاء مكانيكي مي‌باشند) ياري رساند. ALLIANCE يك معماري كنترلي توزيع شده تحمل‌پذيرخرابي براي رباتهاي همكارغير متجانس است كه با انتخاب اعمال به صورت تطبيقي كنترل همكاري بين رباتها را انجام مي‌دهد. در اين معماري، رباتها توانايي انجام تعدادي عمليات سطح بالا كه در يك

---

[1] Middle agent

مأموريت، منجر به انجام وظيفه مي‌شوند را دارا هستند و در هر زمان بايد يك عمل مناسب را با توجه به نيازمنديهاي مأموريت، فعاليت‌هاي ساير رباتها، حالت محيط و حالات دروني ربات انتخاب كنند. انتخاب عمل به صورت تطبيقي با توجه به علايق رباتها وتحليل كارآيي ساير رباتها در زمانهاي گذشته و زمان حال انجام مي‌شود. كارآيي يك ربات با چگونگي تاثير آن بر روي محيط اندازه‌گيري مي‌شود. درALLIANCE ،رباتها با روش مبتني بر رفتار طراحي شده‌اند. در ساختار مبتني بر رفتار، تعدادي رفتار منجر به وظيفه، به صورت همزمان فعال هستند كه هر يك از آنها از سنسورها اطلاعات دريافت كرده و خروجي تعدادي از محركها را كنترل مي‌كنند. رفتارهاي سطوح پايين در ارتباط با اعمال حياتي مهم براي ربات مانند ممانعت از برخورد با موانع هستند ورفتارهاي سطوح بالاتر متناظر با اهداف متعالي‌تر. رفتارهاي سطوح بالا در صورت ضرورت مي‌توانند خاموش‌كنندة[1] خروجي رفتارهاي سطوح پايين‌تر شده و يا بازدارندة[2] خروجي آنها باشند. مي‌توان ادعا كرد مناسب‌ترين راه‌حل براي افزودن قابليت تحمل‌پذيري خرابي به رباتهاي همكار استفاده از يك معماري كنترلي مبتني بررفتار مناسب مي‌باشد. تا كنون معماريALLIANCE موفق‌ترين معماري در اين زمينه بوده است. البته در صورتي مي‌توان از اين معماري استفاده كرد كه وظايف رباتهاي گروه، مستقل از هم بوده و تأثيريكديگر را از بين نبرند كه بسياري از كاربردهاي عملي رباتهاي همكار دراين شرايط صدق نمي‌كنند. در [۲۴] ويژگي‌هاي عمومي وظايف در مأموريت يك گروه از رباتهاي همكار بررسي شده، كه نتيجة آن توصية روش كمك‌رساني رباتها به يكديگر براي افزايش تحمل‌پذيري خرابي مي‌باشد.در اين روش با توسعة معماري ALLIANCE قابليت كمك‌رساني به آن اضافه شده است.

در فرايند كمكي كه در [۲۴] شرح داده شده است، دو نكته بسيار مهم وجود دارد كه در هر معماري كه اين مفهوم را پشتيباني كند بايد مدنظر قرار گيرد. فرض كنيد كه تعدادي ربات در تيمي كه افزونگي (در تعداد يا قابليت‌هاي رباتها) دارد مشغول فعاليت باشند. درخلال فعاليت براي انجام مأموريت گروه يكي از رباتها احساس مي‌كند كه از انجام وظيفة محوله ناتوان است و بنابراين

۱-Suppress

۲-Inhibit

درخواست كمك صادر مي‌كند. هر يك از اعضاء با دريافت درخواست كمك بايد براي پاسخ‌گويي يا عدم پاسخ‌گويي به آن تصميم بگيرد. پس از تصميم‌گيري نيز ربات‌هاي كمك‌دهنده بايد با همكاري يكديگر ربات معيوب را در انجام وظيفه‌اش ياري رسانند. بنابراين مكانيزم انتخاب عمل و هماهنگي بين ربات‌ها در عمليات كمك‌رساني اهميت فراواني دارد. روشي كه ربات‌ها براساس آن درخواست كمك را پردازش كرده و تصميم‌گيري مي‌كنند در كارآيي و عملكرد تيم تأثير بسزايي دارد. پارامترهاي مختلفي وجود دارد كه الگوريتم تصميم‌گيري ربات كمك‌كننده وابسته به آن مي‌باشد، از جمله مي‌توان به موارد زيراشاره كرد:

- فاصله تا رباتي كه درخواست كمك كرده،
- قابليت‌هاي مكانيكي و تجربه و خبرگي خود در مقايسه با ديگران،
- اولويت وظيفه‌اي كه ربات مشغول انجام آن است و وظيفه‌اي كه براي انجام آن درخواست كمك شده است،
- ميزان بحراني بودن وظيفة ربات معيوب،
- وضعيت پيشرفت وظيفة ربات درخواست كننده و وظايف ساير رباتها،
- ميزان تلاشي كه تاكنون براي انجام وظيفه انجام شده است.

معماري ALLIANCE به عنوان پايه‌اي براي انتخاب تطبيقي عمليات انتخاب شده است، ولي با توجه به اينكه اين معماري قابليت پشتيباني كمك را ندارد، در[٢٤] اين قابليت به معماري اضافه شود.

در [۴۷] كه در ادامة معماري ALLIANCE معرفي شده است، معماري تكميل‌يافته‌اي با نام L-ALLIANCE معرفي شده است كه در آن از الگوريتم‌هاي زمانبندي و تقسيم وظيفه در سيستم‌هاي چندپردازنده‌اي، ايده گرفته‌شده است و تقسيم وظيفة اوليه به گونه‌اي انجام شده كه مقداري همپوشاني در قابليت انجام وظيفه ميان رباتها موجود باشد. البته دو ربات مختلف دو وظيفه را با دو كارايي مختلف به انجام مي‌رسانند. مساله‌اي كه پاركر در ارائة اين معماري آن را حل كرده است يعني *انتخاب متناسب وظيفه در يك سيستم مبتني بر رفتار و با احتساب ملزومات يك سيستم تحمل پذير خطا*، يك مسالة NP-Complete است و بنابراين با ارائة راه‌حلهايي ابتكاري سعي شده است كه به راه‌حل بهينه كه رسيدن به آن در عمل غيرممكن است، نزديك شد. رباتهاي اين سيستم چند رباته با داشتن قابليت يادگيري در دو فاز مختلف

فعال[1] و تطبيقي[2] و با فيدبك نمودن كارايي كه آن را مانيتور مي‌كنند،قادر به بهبود عملكرد خود هستند. به اين منظور دو عمل تقسيم مجدد وظيفه[3] و ترتيب‌دهي وظيفه[4] توسط رباتها صورت مي‌گيرد. براي تقسيم مجدد وظيفه، رباتها يكي از سه سياست زير را به كار مي‌بندند:

- Distrust Performance Knowledge about Teammates
- Let the Best Robot Win
- Give Robots a Fighting Chance

در حالي كه براي مسالة ترتيب‌دهي وظيفه، يكي از سه استراتژي كه در زير آورده شده است را به كار مي‌بندند:

- Longest Task First
- Modified Shortest Task First
- Modified Random Task Selection

اين سياست‌ها چنانكه از نامشان نيز پيداست، به ترتيب طولاني‌ترين كار، كوتاهترين كار و كاري را به صورت تصادفي براي انجام دادن انتخاب مي‌كنند. البته پاركر بر اساس آزمايشهايي كه انجام داده است، نتيجه گرفته است كه انتساب وظيفه‌اي كه طولاني‌ترين زمان اجرا را به وسيلة يك ربات دارد به اين رباتها، باعث افزايش نامطلوب زمان پاسخ مي‌گردد و به همين دليل اين سياستها را بهبود داده و آنها را تغييريافته ناميده است. تغييري كه در اين سياستها داده است بدين شكل است كه وظيفه ها را براي هر ربات به دو دسته تقسيم كرده است، به اين شكل:

۱- وظايفي كه يك ربات بهتر (سريعتر) از بقيه قادر به انجام آنهاست.

۲-كل وظايفي كه يك ربات قادر به انجام آنهاست.

سپس پاركر، رباتها را موظف نموده است كه ابتدا از دستة اول با سياست LTF[5] و در صورت عدم وجود وظيفه‌اي در دستة اول از دستة دوم با سياست STF[6] وظيفه انتخاب نمايد و بدين ترتيب به زمان پاسخ مطلوبي دست يافته است.

۳-۸- استفاده از افزونگي سخت افزاري با ايدة سيستم‌هاي بيولوژيكي

در دانشگاه York انگلستان، تحقيقات گسترده‌اي در زمينة استفاده از افزونگي سخت افزاري با ايدة سيستم‌هاي

1 Active
2 Adaptive
3 Task Re-allocation
4 Task Ordering
5 Longest Task First
6 Shortest Task First

بيولوژيكي انجام شده است. در اينجا به شرح يكي از جالب‌ترين‌كارهاي انجام شده در اين زمينه مي‌پردازيم.

كاربردهاي كنترلي بلادرنگ نيازمند تعامل تعداد زيادي از مولفه‌ها هستند و با پيچيده‌ترشدن اين سيستم‌ها، قابليت اطمينان آنها كاهش مي‌يابد و بنابراين به تحمل‌پذيري خطا بايدبه نحوي پرداخته شود كه قابليت اطمينان در حد قابل قبولي باقي بماند. در [۱۳] يك مكانيزم ساختاردهي مجدد به وسيلة روشهايي كه در رشد جنين موجودات زنده استفاده مي‌گردد ارائه شده است. در اين سيستم نشان داده شده است كه ويژگيهاي سطح پايين و با سرعت بالا كه در رفع خطاي سيستم بدن جنين در طي فرايند تكاملي شكل مي‌گيرد، يك روش مناسب براي كاربردهاي كنترلي بلادرنگ است.

مفهوم Embryonics درابتدا در سال ۱۹۹٦ ارائه شد. اين واژه كه از تركيب Embryology و Electronic استخراج شده است، از ايدة‌رشد جنين در ارگانيزمهاي چند مولكولي گرفته شده است. اولين ايده آن است كه يك موجود نوعي از يك تك سلول (Fertilized Egg) رشد يافته است و درطي فرايند تقسيمهاي چندگانه به شكل نهايي رسيده است. بلافاصله پس از توليد موجود فقط يك كپي از DNA موجود است و در طي رشد جنين، يك كپي از آن به هر سلول ارسال مي‌شود. به بيان ديگر قابليت هر سلول يا گروهي از سلول‌ها توسط همسايه‌هاي آنها مشخص مي‌گردد و عملكرد هر سلول در صورتي كه در موقعيت ديگري در بدن جنين قرار بگيرد، تفاوت خواهد نمود. اين مساله به اين شكل امكان پذير مي‌گردد كه هر سلول يك كپي كامل از DNA كه ارگانيزم را توصيف مي‌كند دارا مي‌باشد.

۳-۸-۱- تحمل‌پذيري خطا درآرايه‌اي از پردازنده‌ها[۱]

تحمل‌پذيري خطا در آرايه‌اي از پردازنده‌ها، با ايدة نگاشت آرايهٔ منطقي به آرايهٔ فيزيكي بدون خطا تحقق مي‌يابد. يعني هر سلول منطقي بايد يك سلول فيزيكي متناظر داشته باشد. وقتي خطايي رخ مي‌دهد، يك مكانيزم بايد وجود داشته باشد كه آرايهٔ منطقي را طوري تغيير ساختار بدهد كه توسط سلول هاي فيزيكي بدون خطا نمايش داده شوند. كلية روشهاي ساختاردهي مجدد با استفاده از يكي از دو روش افزونگي سخت افزاري يا زماني تحقق مي يابد.

۱ Processor Arrays

در افزونگي زماني، وظايف سلول‌هاي مواجه شده با خطا ميان همسايه هاي آنها توزيع مي‌شود و در هنگامي كه تغيير ساختار رخ مي‌دهد، پردازنده‌ها زمان خود را ميان انجام وظايف خودشان و وظايف چنين سلول‌هايي تقسيم مي‌كنند و بنابراين سيستم در كل با كاهش كارايي مواجه مي‌شود.

در افزونگي سخت افزاري، سلولهاي اضافه براي جايگزيني با سلول‌هاي خراب استفاده مي‌شود. بنابراين الگوريتم‌هاي جايگزين براي استفاده از اين منابع بايد بهينه شوند. در حالت ايده‌آل يك آرايه N تايي از پردازنده ها قادر خواهند بود كه خرابي N سلول را برطرف نمايد ولي در عمل به علت محدوديت هاي موجود، اين ميزان قابل تحقق نخواهد بود.

اكثر الگوريتم‌هاي ساختاردهي مجدد كه مبتني بر افزونگي سخت افزاري هستند به الگوريتم‌هاي پيچيده‌اي متكي هستند كه منابع منطقي را به عناصر فيزيكي مي‌نگارند. در اغلب اين الگوريتم‌ها، يك پردازندة مركزي عمليات تشخيص خطا را به عهده دارد.

اين روشها با وجود اينكه كارا هستند ولي به دليل ذات متمركزي كه دارند، از قابليت اطمينان كمي برخوردارند ،چرا كه در صورت مواجهه با خطاي همان يك پردازنده به راحتي دچار مشكل مي‌شوند.

يك روش جايگزين، توزيع الگوريتم تشخيص و ساختاردهي مجدد ميان همة سلول هاي آرايه است. در اين روش هيچ عامل مركزي لازم نيست و بنابراين قابليت اطمينان افزايش يافته ، پاسخ زماني سيستم بهبود خواهد يافت. اگرچه اين روش غير متمركز، داراي پيچيدگي بيشتري است و رد و بدل اطلاعات بيشتري را هم مي‌طلبد.

۳-۸-۲- اعمال ايدة تحمل‌پذيري خطا به شيوة سيستم‌هاي بيولوژيكي

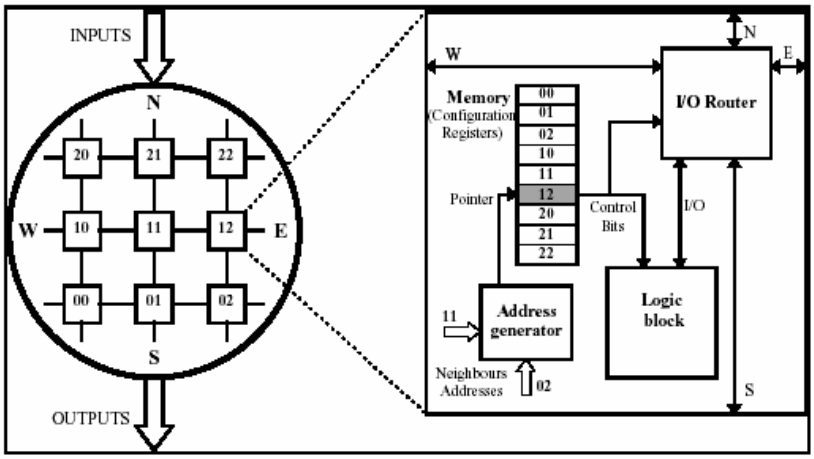
علم Embryonic از بررسي فرايندهاي بيولوژي مولكولي سرچشمه مي‌گيرد، بنابراين با پذيرفتن ويژگي سلول‌هاي موجودات زنده و با تبديل آنها به آرايه‌هاي دوبعدي از سلول‌ها، نشان داده مي‌شود كه ويژگي‌هايي كه مختص موجودات زنده هستند مانند توليد مثل و رفع عيب توسط خود[۱] ،

[۱] Self-Repair

قابل اعمال به مدارهاي مجتمع مي‌باشند. شكل 10 مولفة اصلي سيستم هاي Embryonic را نشان مي‌دهد.

هر سلول مختصات خود را از همسايه هاي نزديك جنوب و غرب خود دريافت مي‌كند. واحد توليد آدرس، سطر و ستون مختصات همسايهٔ شمال و شرق را محاسبه مي‌كند. بلاك منطقي[1] به وسيلهٔ يك رجيستر ساختاردهي[2] كنترل مي‌شود كه به وسيلهٔ مختصات انتخاب مي‌شود. هنگامي كه يك خطا تشخيص داده شد، محاسبهٔ مختصات توسط سلول ديگر با مبادلهٔ مختصات آن و بنابراين عملكردش آغاز مي‌گردد. ساختار نشان داده شده داراي برتري‌هاي قابل‌ملاحظه اي است:

- ✓ ساختار بسيار رايجي است كه قابليت پياده‌سازي روي سيلسكون را دارد.
- ✓ عملكرد بلاك منطقي مستقل از عملكرد بقية بلاك هاست و بدون تغيير در بقية اجزا قابليت جايگزيني و تغيير را دارد.
- ✓ بنابراين، قابليت سريع تغيير ساختار در اين طرح، آن را براي استفاده در كاربردهاي بلادرنگ كنترلي مناسب مي‌سازد [۱۳] [۱۴] .
- ✓



شكل 10 - مولفة اصلي سيستم هاي Embryonic

۳-۹- نتيجه‌گيري

در اين فصل بخش عمده اي از تحقيقات انجام شده در زمينة تحمل‌پذيري خرابي در سيستم هاي گسترده بررسي شدند. با وجود گستردگي اين تحقيقات هنوز روشي عمومي براي افزودن قابليت

[1] Logic Block

[2] Configuration Register

تحمل‌پذيري‌خرابي به سيستم‌هاي توزيع‌شده ارائه نشده است. تعدادي از مقالات راه‌حل‌هايي خاص‌منظوره ارائه کرده‌اند که با وجود حصول موفقيت در مسائل عنوان‌شده، در ساير موارد قابل استفاده نيستند. بنابراين با وجود اينکه سيستم‌هاي چندعامله قابليت دارا شدن خاصيت تحمل‌پذيري‌خطا به کمک روشهاي رايجي مانند افزونگي را دارند، اما از بين کلية روش‌هاي قابل استفاده، ايدة پيکربندي مجدد و تغيير ساختار سيستم در شرايط بروز خطا مناسب‌ترين و جديدترين روش به نظر مي‌رسد. فلسفة طراحي بر اساس پيکربندي مجدد[1] آن است که:

هر عامل در صورت رخداد خطا براي يکي از هم‌تيمي‌هايش و تصميم به کمک به او، نقش جديدي را نيز تقبل نمايد و با انجام وظيفة وي سعي در حفظ کارايي کلي سيستم مي‌نمايد.

اين تصميم‌گيري گسترده براي قبول يا رد درخواست کمک، انتخاب يک کمک‌رسان مناسب و تعيين بهترين ترتيب کمک‌رساني از مهم‌ترين مسائلي است که در اين تحقيق سعي شده که روشهايي عملي براي حل آنها ارائه گردد.

روشهايي که در اين تحقيق براي حل مساله در پيش گرفته‌ايم، بيشترين شباهت را به روشهاي مبتني بر الگوريتم‌هاي زمانبندي و تقسيم وظيفه در سيستم‌هاي چندپردازنده‌اي و بنابراين در ايده‌هاي کلي به [٤٧] دارد. اما به دليل چند تفاوت عملي در فرضيات، انتظارات و اهدافي که ميان سيستم پيشنهادي در اينجا و سيستم L-ALLIANCE وجود دارد، تفاوتهايي در اين طراحي پيشنهادي و طراحي پاركر به چشم مي‌خورد. به طور دقيق‌تر، هدفي که در اين تحقيق دنبال مي‌شود، مي‌نيمم نمودن تعداد عامل‌هاي درگير براي انجام ماکزيمم تعداد وظيفه‌هايي است که به عاملها منتسب مي‌شود. يعني همواره مطلوب آن است که اگر انجام m وظيفه توسط n1 و n2 عامل کمک‌رسان امکان‌پذير است و n1 از n2 کمتر است، زمانبندي که n1 عامل را دخيل مي‌نمايد، انتخاب نماييم. بنابراين با وجود اينکه به دليل توالي انجام وظيفه‌ها در سياستي که انتخاب مي‌کنيم، احتمالا زمان پاسخ طولاني‌تر مي‌شود، اما به دليل باقي گذاشتن تعداد بيشتري عامل به صورت رزرو براي آينده، اين سياست بر ديگري برتري دارد. به علاوه در صورتي که تعداد خرابي‌ها زياد بوده و سيستم شديدا نيازمند کمک‌رساني باشد، برتري اين

1 Reconfiguration

سياست بر سياستهايي كه به زمان پاسخ بهاي بيشتري مي‌دهند، به وضوح قابل لمس است.

از طرف‌ديگر روش پيشنهادي اين تحقيق، به هر نوع خطايي (اعم از كوتاه‌مدت، بلندمدت و متناوب) كه براي هر يك از عاملها بروز نمايد، مي‌پردازد و اختصاصي يك يا چند عامل خاص مانند عاملهاي واسط ندارد و اين يك برتري بر روشي است كه در [۸] [۹] مطرح شده است.

به عنوان مقايسه‌اي اجمالي با ديگر روشهايي كه در اينجا معرفي شدند، لازم به توضيح است كه روش پيشنهادي اين تز مبتني بر هيچ عامل افزونه‌اي مانند عامل ميانی و محافظ نبوده از اين لحاظ با روشهاي مبتني بر اين چنين عاملهايي كه در بالا به تفصيل معرفي شدند، تفاوت عمده‌اي دارد. به علاوه در اين تحقيق فرض بر آن است كه تشخيص خطا توسط هر عامل صورت مي‌گيرد و نيازي به مولفة تشخيص خطا وجود ندارد و اين هم يك تفاوت ديگر با سيستم‌هايي است كه صرفا به روشهاي تشخيص خطا توسط عامل‌ها پرداخته‌اند.

٤- الگوريتم‌هاي زمانبندي وظيفه در سيستم‌هاي عامل چندپردازه‌اي بلادرنگ

در اين فصل ابتدا به صورت گذرا، چند تعريف كلي از زمانبندي در سيستم‌هاي‌عامل بيان مي‌شود، سپس چند تحقيق مرتبط به اين زمينه مرور مي‌گردد و ايدة استفاده از الگوريتم‌هاي زمانبندي بلادرنگ و چندپردازنده‌اي در سيستمهاي چندعامله بيان خواهد شد.

٤-١- مقدمه [٣٤]

زمانبندي به معناي تخصيص منابع و زمان به وظيفه‌هاست به نحوي كه تابع كارايي خاصي بهينه گردد. اهداف زمانبندي معمولي عبارتند از: انصاف ميان تخصيص زمان پردازنده به فرآيندها، عدم گرسنگي فرآيندها، استفادة كارآمد از وقت پردازنده‌ها و كمي سربار. در حالي كه از يك زمانبند بلادرنگ انتظار مي‌رود كه ملزومات عملكرد بلادرنگ را فراهم سازد. زيرا رعايت محدوديت‌هاي زماني شروع و پايان وظيفه‌ها يا فرآيندها، مقدم بر انصاف يا اولويت است.

زمانبندي داراي سه نوع بلندمدت، ميان‌مدت و كوتاه‌مدت است:

زمانبندي بلندمدت : تصميم‌گيري در مورد افزودن به مجموعه‌اي از فرآيندها براي اجراست .

زمانبندي ميان‌مدت: تصميم‌گيري در مورد افزودن به تعداد فرآيندهايي كه بخشي يا تمام آنها در حافظة اصلي است، مي‌باشد.

زمانبندي كوتاه‌مدت: تصميم‌گيري در مورد اينكه كدام فرآيند موجود در حافظة اصلي براي اجرا توسط پردازنده انتخاب شود،را بيان مي‌كند.

٤-٢- علت استفاده از ايده‌هاي زمانبندي سيستم‌هاي عامل در اين تحقيق

ما در اين تحقيق، با مساله‌اي مشابه زمانبندي كوتاه‌مدت روبرو هستيم، چرا كه در لحظاتي كه در سيستم خرابي رخ مي‌دهد عاملها بايد از ميان درخواستهاي كمك، انتخاب نمايند كه زمان و حافظة خود را صرف كمك‌رساني به كدامين عامل يا عاملها بنمايند. بنابراين اكنون به الگوريتمهاي مربوط به زمانبندي چندپردازنده‌اي بلادرنگ

مي‌پردازيم كه با سيستمهاي چندعامله كه مورد بحث اين پروژه است، شباهتهاي عمده‌اي دارند.

سيستمهاي چندعامله از اين لحاظ كه در هر لحظه كلية عاملها با يكديگر فعال بوده و به انجام تعامل با يكديگر مشغولند، شبيه سيستمهاي چندپردازنده‌اي مي‌باشند و در صورتي كه عامل‌ها داراي محدوديت‌هاي زماني اجرا نيز باشند، با زمانبندي چندپردازنده‌اي بلادرنگ مواجه خواهيم بود. اين مساله موجب مي‌شود كه الگوريتم‌ها و روش‌هاي زمانبندي و توزيع وقت پردازنده ميان فرآيندها، قابل اعمال به سيستم‌هاي چندعامله باشد. البته از آنجا كه عمليات تخصيص وقت پردازنده به فرآيندها در سيستمهاي چندپردازنده‌اي توسط سيستم‌عامل و به صورت مركزي صورت مي‌گيرد و در سيستمهاي چندعامله چنين جايگزيني وجود ندارد، الگوريتم‌ها پيش از انتقال به سيستم‌هاي چند عامله نياز به اصلاحاتي دارد كه مهم‌ترين آنها تبديل نمودن به صورت توزيع‌شده مي‌باشد. بنابراين در اين فصل اصول و ايده‌هايي كه از سيستم‌هاي چندپردازنده‌اي گرفته شده تا به سيستمهاي چندعامله اعمال گردند، بيان مي‌شوند و در فصل ٦ كه به بيان روشهاي اصلي اين تحقيق مي‌پردازد، نحوة استفاده از اين ايده‌ها بيان مي‌گردد.

٤-٣- ويژگي‌هاي سيستم‌هاي بلادرنگ[٣٨]

سيستم بلادرنگ به سيستمي اتلاق مي‌شود كه در آن صحت سيستم نه تنها به صحت نتايج محاسباتي، بلكه به زماني كه خروجي توليد مي‌شود نيز وابسته باشد، مانند سيستمهاي كنترل فرآيند و كنترل پرواز. تعداد زيادي از سيستم‌هاي امروزي به صورت از پيش‌دانسته‌اي بر اساس محدوديت‌هاي زماني وظيفه‌هاي شناخته شدة سيستم، برنامه‌ريزي مي‌كنند و بنابراين داراي زمانبندي استاتيك هستند. البته زمانبندي استاتيك مشكلات خاص خود را از قبيل عدم انعطاف‌پذيري و هزينة بالا دارد.

نسل بعدي، سيستمهاي بلادرنگ سخت هستند كه بايد به صورت ديناميك، قابل پيش‌بيني و قابل انعطاف طراحي شوند و الگوريتمهاي زمانبندي در چنين سيستمهايي بايد محدوديتهاي زماني آنها را در نظر بگيرند.

٤-٤- زمانبندي بلادرنگ و معيارهاي ارزيابي كارايي آن[٢٦]

زمانبندي، مهمترين مسالة روز در سيستم‌هاي بلادرنگ است. معيارهاي ارزيابي كارايي زمانبندي غالبا به دامنة كاربرد بستگي دارند ولي معمولا وظيفه‌ها با پارامترهايي از قبيل: زمان لازم براي تكميل، منابع مورد نياز، سطح اهميت، نيازهاي اولويتي، نيازهاي ارتباطي و البته محدوديت‌هاي زماني مشخص مي‌شوند. اگر يك وظيفه پريوديك باشد،پريود آن مهم است و اگر غير پريوديك باشد مهلت زماني آن مهم مي‌شود. هر دو نوع وظيفة پريوديك و غيرپريوديك، محدوديت‌هاي زماني شروع كار را دارند.

الگوريتمهاي مختلف زمانبندي در سيستم‌هاي بلادرنگ به چهار دستة كلي زير تقسيم مي‌شوند:

روش‌هاي استاتيك مبتني بر جدول[1]: اين روشها بر اين اساس استوارند كه منابعي كه براي فراهم نمودن مهلت زماني وظيفه‌هاي بحراني لازم هستند بايد از پيش در اختيار گرفته شوند تا بتوانند به صورت قطعي و تضمين‌شده‌اي در زمان لازم در اختيار باشند. اين وظيفه‌ها معمولا به صورت استاتيك زمانبندي مي‌شوند، به طوري كه حتي در بدترين شرايط، مهلت‌زماني قابل‌فراهم‌نمودن باشد. اگر وظيفه‌ها داراي ويژگي‌هاي ساده‌اي باشند، مي‌توان يك جدول بر اساس EDF[2] يا SJF[3] تهيه نمود اما در حالت پيچيده‌تركه وظيفه‌ها داراي تقدم يا تاخر، ارتباط و ... باشند با مسالة NP-Complete روبرو خواهيم بود كه در زمانبندي مشكل‌زا خواهد بود و بايد از روشهاي ابتكاري[4] بهره بگيريم. اغلب الگوريتمها از روش Branch-and-Bound استفاده مي‌كنند تا يك زمانبندي قابل‌اجرا بيابند.

روش‌هاي استاتيك مبتني بر اولويت و به صورت غير انحصاري[5]: اين روش زمانبندي بيشتر در سيستم‌هاي اشتراك زماني استفاده مي‌شود. در سيستمهاي غيربلادرنگ، اولويت يك وظيفه بر اين اساس كه وابسته به CPU يا I/O است، تغيير مي‌كند. در حالي كه در سيستم‌هاي بلادرنگ، انتساب اولويت وابسته به محدوديت‌هاي زماني يك وظيفه است و اين انتساب

1 Static Table Driven Approaches
2 Earliest Deadline First
3 Shortest Job First
4 Heuristic
5 Static Priority-driven Preemptive Approaches

مي‌تواند ديناميك يا استاتيك باشد. انتساب اولويت به شكل ايستا از اين لحاظ قابل‌توجه است كه به محض رسيدن يك وظيفه، اولويت آن معلوم مي‌شود و با گذشت زمان اين اولويت مجددا محاسبه نمي‌شود، در حالي كه در انتساب اولويت به شكل پويا، اولويت يك وظيفه وقتي يك وظيفة جديدتر با مهلت زماني كوتاهتر مي‌رسد، ممكن است تغيير نمايد.

روش‌هاي ديناميك مبتني بر برنامه‌ريزي[1]: اين روش مبتني است بر بررسي‌هايي به شكل ديناميك براي تشخيص قابل انجام بودن زمانبندي‌ها. بدين معنا كه يك وظيفه با ساختن نقشه‌اي براي اجراي آن در صورتي كه همة ديگر وظيفه‌هايي كه تضمين شده‌اند،به مهلت زماني خود پايدار بمانند، ضمانت اجرايي مي‌يابد. يك الگوريتم ضمانت، كليه موارد از جمله زمان اجرايي در بدترين حالت، نيازمندي‌ها به منابع، محدوديت‌هاي زماني، محدوديت‌هاي تقدم و تاخر را در نظر مي‌گيرد. در يك سيستم كنترل گسترده، وقتي يك وظيفه به يك گره مي‌رسد، زمانبند موجود در آن گره تلاش مي‌نمايد كه آن وظيفه را پيش از سرآمدن مهلت زماني‌اش، از لحاظ اجرايي تضمين نمايد. اگر اين تلاش به ثمر نرسد، مولفه‌هاي زمانبندي موجود در گره‌هاي ديگر همكاري مي‌نمايند تا مشخص كنند كه كدام گره منابع كافي و در عين حال اضافه دارد كه اين وظيفه‌ها را به صورت تضمين شده‌اي تقبل نمايد. طبيعي است كه الگوريتم‌هاي ديناميكي كه به صورت از پيش‌دانسته، زمان‌رسيد وظيفه‌ها را نمي‌دانند، نمي‌توانند كارايي بهينه را تضمين نمايند. اما از ميان اين الگوريتم‌هاي پويا، الگوريتمي كه تعداد بيشتري وظيفة قابل‌اجرا را تضمين نمايد، الگوريتم بهتري تلقي مي‌شود.

روش‌هاي ديناميك مبتني بر بهترين تلاش[2]: اين دسته روش‌هاي زمانبندي در بيشتر سيستم‌هاي بلادرنگ امروزي استفاده مي‌شوند. در اين چنين سيستم‌هايي، يك اولويت براي هر وظيفه بر اساس ويژگي‌هاي آن محاسبه مي‌شود و بر اساس همين اولويت‌ها زمانبندي مناسب صورت مي‌گيرد. در اين زمينه اغلب، الگوريتم‌هاي EDF وLLF[3] تا زماني كه پركاري[4] رخ نداده است داراي عملكرد اپتيمال هستند. اما در صورتي كه در سيستم پركاري رخ بدهد، با توجه به اينكه

1 Dynamic Planning-based Approaches
2 Dynamic Best-Effort Approaches
3 Least Laxity First
4 Overloading

الگوريتم‌هاي پويا در اين شرايط بهترين عملكرد را دارند، وظيفة بعدي براي اجرا يا حذف از اجرا، بايد با دقت انتخاب شود. روش‌هاي مبتني بر بهترين تلاش سعي مي‌كنند كه مجموع وظيفه‌هاي تكميل‌شده در شرايط پركاري را ماكزيمم نمايند. بنابراين از الگوريتم‌هاي مبتني بر اولويت و به شكل غير انحصاري استفاده مي‌كنند.

براي اين منظور از هر يك از روش‌هاي EDF، SJF، LLF و FCFS[1] انتخاب وظيفة بعدي به شكل تصادفي مي‌توان استفاده نمود. مهم‌ترين مشكل اين دسته روش‌هاي زمانبندي، عدم قابليت‌پيش‌بيني آنها و اپتيمال بودن نسبي آنهاست. يك الگوريتم ديناميك براي زمانبندي، هنگامي اپتيمال ناميده مي‌شود كه همواره يك زمانبندي قابل انجام ارائه نمايد و اين تنها در شرايطي كه يك الگوريتم كلية اطلاعات را از پيش در اختيار داشته باشد، امكان‌پذير است كه چنين فرضي خود در عمل غير ممكن است و اين مساله موجب مي شود كه در مورد اغلب مسائل جهان واقعي چنين الگوريتم پويا و اپتيمالي وجود نداشته باشد.

٤-٥- موارد مهم ديگر در مورد زمانبندي

در مورد زمانبندي در يك سيستم بلادرنگ، موارد مهم ديگري هم وجود دارد كه با توجه به آنها مي‌توان كيفيت زمانبندي را بهبود داد، اين موارد عبارتند از:

٤-٥-١- پشتيباني از تحمل‌پذيري خطا

در[٢٧] يك مكانيزم مهلت زماني معرفي شده كه قادر است تضمين نمايد كه يك وظيفة اصلي مي‌تواند در صورتي كه هيچ خرابي رخ ندهد، به مهلت زماني خود پايدار بماند و نيز در صورتي كه خرابي رخ بدهد يك وظيفة جايگزين (با دقت كمتر) در حوالي مهلت زماني خود تكميل شود. اگر وظيفة اصلي اجرا شود نيازي به اجراي وظيفة جايگزين نيست و زمان باقيمانده صرف انجام وظيفه‌اي نمي‌شود. اين روش فقط در مورد كارهاي پريوديك است و همة وظيفه‌ها را نيز به صورت غيرانحصاري فرض مي‌كند. در اين روش محاسبات به صورت از پيش‌انجام‌شده‌اي در مورد زمانبندي‌هاي اصلي و جايگزين انجام مي‌شود و به صورت جدولي ذخيره مي‌شود. در [٢٨] روشي معرفي شده است كه قادر است به سرعت در زمان

1First Come First Served

رخ دادن خرابي از يك وظيفه به وظيفة جديد ديگري سوئيچ نمايد، كه البته اين وظيفه از پيش تعيين شده است. اين روش مي‌تواند تضمين نمايد كه مهلت‌هاي زماني سخت حتي در مواجهه با ماكزيمم تعداد خرابي‌ها، قابل برآورده‌نمودن هستند.

زمانبندي‌هاي پيش‌بيني‌شدة ثانويه فقط در صورت بروز خطا اجرا مي‌شوند. اگرچه چون زمانبندي‌هاي ثانويه نياز به صرف زمان پردازشي دارند تا در موردشان تصميم‌گيري شود، كارايي زمانبندي‌هاي اوليه و اصلي را كمي كاهش مي‌دهند كه اين در اصل همان بهايي است كه براي محاسبات اندكي كه به صورت Online در شرايط بروز خطا اجرا مي‌شوند، بايد بپردازيم.

اين دو روش زمانبندي كه با توجه به مسالة تحمل پذيري خطا عمل مي‌كنند براي سيستم هاي استاتيك و Embedded كه در آنها تحمل‌پذيري خرابي مسالة بسيار مهمي است و مهلت‌هاي زماني بسيار حساس دارند، مناسب هستند. در اين چنين مواردي، ميزان كارايي CPU اهميت چنداني ندارد، بلكه در عوض تعويض نمودن وظيفه‌هاي اصلي و ثانويه مهم مي‌باشد.

٤-٥-٢- استفادة بهينه از زمانهاي باقيمانده از اجراي وظيفه‌هاي اصلي

واريانس در زمان‌هاي پيش‌بيني شده براي اجراي وظيفه‌ها ممكن است بعضا موجب شود كه برخي وظيفه‌ها پيش از زمان پيش‌بيني شده توسط زمانبند، تكميل گردند. در اين موارد Task Dispatcher مي‌تواند زمان باقيمانده از تكميل زودتر را Reclaim نموده و به مصرف تكميل ديگر وظيفه‌ها برساند. بديهي است كه وظيفه‌هاي غيربلادرنگ مي‌توانند در زمان‌هاي باقيمانده اجرا شوند ولي ارزشمندتر از اين مساله آن است كه وظيفه‌هاي بلادرنگ با محدوديت‌هاي زماني را به صورت تضمين شده‌اي در اين زمانهاي خالي به اجرا درآوريم. وقتي زمان واقعي اجراي يك وظيفه با زمان اجراي بدترين حالت آن متفاوت است، آنومالي زمان اجرا رخ مي‌دهد و اين آنومالي‌ها مي‌توانند موجب شود كه برخي وظيفه‌ها مهلت زماني مقرر خود را از دست بدهند. [٢٦]

٤-٦- مروري بر چند الگوريتم زمانبندي در سيستم‌هاي بلادرنگ

اكنون به چند نمونه از تحقيقاتي كه در زمينة زمانبندي در سيستم‌هاي چندپردازندة بلادرنگ انجام شده است، مي‌پردازيم.

[٣٢] به توسعة يك الگوريتم زمانبندي وظيفه براي سيستم‌هاي گسترده پرداخته است كه هدف آن زمانبندي گستردة تطبيقي و انعطاف‌پذير وظيفه‌هاي بلادرنگ است. چرا كه سيستم‌هاي بلادرنگ و مدرن امروزي به توان محاسباتي زيادي نياز دارند و سيستم‌هاي تك‌پردازنده‌اي غالبا مناسب اينكار نيستند. در اين روش يك الگوريتم زمانبندي پوياي بلادرنگ نرم با نام DDSCHED ارائه شده است كه داراي دو مولفة اصلي است :يك زمانبند محلي و يك شماي توزيع شدة زمانبندي. زمانبند محلي بر اساس EDFو مفهوم پنجره طراحي شده است. براي پياده‌سازي از يك ساختمان داده‌اي استفاده شده است كه پيچيدگي زماني را از $O(n^2)$براي EDF به O(n.logn) برساند. شماي گستردة زمانبندي بر اساس الگوريتم‌هاي Bidding استوار است.

روش ديگر بر اين اساس استوار است كه زمانبندي چند كپي از وظيفه‌ها روي پردازنده‌هاي مختلف مي‌توانند موجب افزايش تحمل‌پذيري خطا گردد. يكي از مدلهايي كه براي زمانبندي تحمل‌پذير خرابي در مورد وظيفه‌هاي بلادرنگ وجود دارد، مدل PB (Primary Backup) است. در اين مدل، دو كپي به صورت سريال روي دو پردازندة جداگانه اجرا مي‌شوند و يك آزمون براي بررسي قابل‌قبول بودن[1] نيز روي نتايج انجام مي‌گيرد. نسخة پشتيبان فقط در صورتي اجرا مي شود كه نسخة اصلي در آزمون قابل‌قبول بودن رد شود يا يك خرابي در نرم‌افزار يا پردازنده رخ بدهد. در [٢٩] روشي ارائه شده است كه بر همين اساس عمل مي‌نمايد.

از سوي ديگر وقتي با سيستم‌هاي چندپردازنده‌اي سروكار داريم، تحمل‌پذيري خطا به مراتب مهم‌تر مي‌شود چرا كه با مولفه‌هاي بيشتري مواجه هستيم كه ممكن است دچار خرابي شوند. در[٣١] هم يك الگوريتم زمانبندي براي افزايش تحمل‌پذيري خطا ارائه شده است كه بر اساس Primary / Backup Tasks, Backup Overloading, Backup Deallocation استوار است. Primary / Backup Tasksكه همان مدل ساده‌اي است كه در بالا شرح داده شد ولي

1 Acceptance Test

مفهوم Backup Overloading يعني ممكن است بيش از يك وظيفه در هر بازة زماني زمانبندي گردند. مفهوم BackupDeallocation يعني Reclaim نمودن منابع استفاده نشده با وظيفه‌هاي Backup در حالت انجام عمليات تحمل‌پذيري خطا. اين روش قادر است كه خرابي در هر پردازنده را در هر لحظه تحمل نمايد خواه گذرا باشد يا ماندگار. نتايج شبيه‌سازي نشان مي‌دهد كه به خاطر افزودن توانايي تحمل‌پذيري خطا، مقدار كمي كاهش در امكان زمانبندي وظايف ايجاد مي‌شود.

به عنوان نمونه‌اي ديگر[۳۰] يك شماي زمانبندي را معرفي مي‌كند كه Recovery از چند خرابي با وجود محدوديت‌هاي زماني بلادرنگ سخت را در سيستم‌هاي تك‌پردازنده‌اي در نظر مي‌گيرد. با در نظر گرفتن زمانبندي EDF براي وظيفه‌هاي غيرپريوديك و غير انحصاري، يك الگوريتم امكان‌سنجي براي زمانبندي با تحمل‌پذيري خطا با پيچيدگي $O(n^2.k)$ كه n تعداد وظيفه‌هايي است كه قرار است زمانبندي شوند و k ماكزيمم تعداد خرابي‌ها است، ارائه شده است.

و بالاخره [۳۳] يك زمانبندي مبتني بر ارزش[1] است كه توسط آن كاهش قابل قبول كارايي به وسيلة اجرا نمودن وظيفه‌هاي بحراني كه بيشترين ارزش / سود / پاداش[2] را نصيب سيستم مي‌نمايند تامين مي‌شود. در اين روش، هر وظيفه با يك پاداش و تنبيه[3] كه به سيستم بر اساس اينكه مهلت زماني وظيفه را حفظ نموده است يا خير، داده مي‌شود. بعضي از اين روشها يك معيار كارايي تعريف مي‌كنند كه نه تنها پارامترهاي پاداش/ تنبيه را تعريف مي‌كند، بلكه مصالحة ميان قابليت زمانبندي و قابليت اعتماد را هم انجام مي‌دهد. در اين روش يك زمانبندي پويا براي سيستم‌هاي چندپردازنده‌اي بلادرنگ ارائه شده است كه به صورت آگاه از قابليت اطمينان و مبتني بر ارزش[4] عمل مي‌نمايد كه هدف آن ماكزيمم نمودن كارايي است. اين زمانبند، يك سطح از افزونگي براي هر وظيفه انتخاب مي‌كند تا كارايي سيستم را در صورت بروز خطا براي وظيفة اصلي افزايش دهد.

1 Value-based

2 Reward/ Benefit/ Value

3 Penalty

4 Reliability-aware, Value-based Scheduler for Dynamic Multi-processor Real-time Systems

۵- شرح بستر آزمون و فرضيات

در فصل‌هاي قبلي مقدمات و زمينه‌هاي تحقيقاتي مرتبط براي طراحي يك سيستم چندعامله با قابليت تحمل‌پذيري‌خطا، ارائه شد. در اين فصل به تشريح جزئيات طراحي سيستمي كه براي تحقق ايده‌هاي اين پروژه در نظر گرفته‌شده‌اند، مي‌پردازيم و مولفه‌هاي موجود را با جزئيات كافي توضيح مي‌دهيم.

۵-۱- مقدمه‌اي در مورد زبان توصيف طرح

به منظور هماهنگ نمودن ايده‌ها با سيستم‌هاي واقعي سخت‌افزاري و ايجاد امكان سنتز و بررسي ايده‌ها در عمل، زبان سخت‌افزاري VHDL براي توصيف طرح انتخاب شده است. زبان VHDL علاوه بر بهره‌گيري از امكانات رايج هر زبان رايج برنامه نويسي، به دليل اختصاصي كه به مدل‌هاي سخت‌افزاري دارد، داراي ويژگي‌هاي خوبي از قبيل همزماني در اجرا بدون نياز به انجام تمهيدات نرم افزاري براي متوقف نمودن زمان در يك بازة موردنظر، قابليت اعمال كنترل دقيق زماني بر روي رخدادها در هر لحظه و در نهايت نزديك بودن به مرحلة ساخت و سنتز از طريق ابزارهاي رايج سنتز مي‌باشد. البته سيستم طراحي شده قابليت توصيف با هر يك از زبانهاي رايج توصيف نرم افزار به صورت شيءگرا يا عاملگرا را دارد، اما علت انتخاب يك زبان سخت‌افزاري براي توصيف سيستم، تمايل به نزديك بودن زبان توصيف ايده‌ها به زباني است كه با يك ابزار سنتز و به صورت مكانيزه به راحتي به يك سخت‌افزار واقعي نگاشته شود.

۵-۲- شرح كلي سيستم

سيستمي كه براي تحقق ايده‌هاي اين پروژه برگزيده شده است، شبيه يك سيستم كنترل گسترده (DCS) ولي بسيار ساده‌تر است. علت انتخاب اين سيستم گسترده براي چنين منظوري آن است كه يك سيستم كنترل گسترده به علت داشتن عناصر مختلف و مستقل و قابليت تعريف انواع وظيفه‌ها براي عاملها، نمونة خوبي از يك سيستم توزيع‌شده در حالت كلي است. البته انتخاب اين سيستم از كليت طرح و ايده‌ها نمي‌كاهد، چرا كه تنها براي تحقق و بررسي عملي بودن ايده‌ها از اين سيستم استفاده شده است و وابستگي مفهومي ميان ايده‌هاي

پيشنهادي در حالت كلي با اين بستر آزمون وجود ندارد. بلكه طراح هر سيستم گسترده با هر پروتكل ارتباطي خاص و با توصيف سيستم خودش به هر روش استاندارد موجود، تنها با در نظر داشتن اين ايده‌ها در زمان طراحي، مي‌تواند تحمل‌پذيري خطا را به ويژگي‌هاي ذاتي سيستم خود بيفزايد.

در اين سيستم هر عامل به منزله يك كنترلر است. اين سيستم با استفاده از خروجي عامل‌ها كه در اصل فرمان‌هاي كنترلي هستند كه عامل در نتيجه انجام وظيفه محوله، آنها را توليد و صادر نموده است، كنترل قسمتهاي ديگري از plant را بر عهده دارد. مي‌توان فرض نمود كه داده‌هاي مورد نياز هر عامل توسط مولفة Frame Generator از سنسورهاي موجود در محيط جمع‌آوري مي‌شود و به شكل بسته‌هاي برچسب‌خورده با نام آنها در اختيارشان قرار مي‌گيرد. در اصل عامل‌ها با دسترسي به باس مشترك ورودي دادة مورد نظر خود را برمي‌دارند و پس از انجام وظيفة موردنظر، نتيجة خروجي خود را بر روي باس خروجي قرار مي‌دهند. آنچه شرح داده شد، عملكرد سيستم در حالت عادي است. در صورتي كه خرابي در سيستم رخ بدهد (يعني يك عامل با خطايي مواجه گردد)، عامل نيازمند به كمك از طريق باس مشترك كمك، اين مساله را به اطلاع بقيه مي رساند، به عبارتي پرچم نشان‌دهندة وضعيت نياز به كمك را Set مي‌نمايد و از باس مشترك ورودي سيستم قطع مي شود. در صورتي كه عاملي خود را واجد شرايط كمك‌رساني بيابد (كه در مورد مساله واجد شرايط بودن به طور كامل در فصل ٦ بحث خواهد شد) با Set نمودن پرچم ديگري براي نشان دادن اينكه به اين عامل كمك شده است، كمك‌رساني به وي را آغاز مي‌كند. اين پرچم مختص اين مساله است كه در مورد دادة فعلي به عامل نيازمند كمك‌رساني شده است و نيازي به كمك ديگر عاملها نيست. هنگامي كه دادة جديدي توسط Frame Generator براي اين عامل توليد شود، اين پرچم مجددا Reset مي‌شود تا امكان كمك‌رساني مجدد را فراهم نمايد.

٥-٣- شرح وظايف سيستم

در اين سيستم، هر عامل در حالت عادي انجام يك وظيفه را بر عهده دارد. البته وظيفه ها به صورت پريوديك به عاملها محول مي شوند. يك وظيفه در صورتي كه يك عامل وظيفه‌اي كه به او محول شده است را پيش از به سر رسيدن مهلت زماني‌اش كه اندكي جلوتر شرح داده مي‌شود، به

انجام برساند و نتيجة خروجي را بر روي باس موردنظر قرار دهد، اين وظيفه به طور موفقيت‌آميزي به پايان رسيده است. اينكه كدام عامل مسئول انجام كدام وظيفه است، توسط ID كه بر روي بستة ورودي توسط Frame Generator قرار گرفته است، مشخص مي‌شود. بدين ترتيب هر عامل وظيفة حالت عادي خود را انجام مي‌دهد و به دليل وجود مقداري افزونگي ضمني در قابليتهاي عامل در زمان طراحي، عامل بنا به توان پردازشي كه دارد مي تواند كه در صورت لزوم و اگر امكان آن را داشته باشد (كه در اين مورد در فصل ٦ مفصلا صحبت خواهد شد) براي كمك‌رساني اقدام نمايد. هر وظيفه به طور مستقل داراي يك سري ويژگي‌هاست و زماني كه به يك عامل منسوب مي‌شود، ويژگي‌هاي ديگري نيز به آن اضافه مي‌شود كه تمامي آنها در زير آمده است:

١- حجم متوسط پردازشي(Average Amount of Work) : ميزان كاري كه در حالت عادي به ازاء اين وظيفه در مقايسه با ساير وظيفه‌ها، هرعامل بايد انجام بدهد، توسط اين عدد نمايش داده مي‌شود. اين پارامتر واحد ندارد و صرفا يك كميت نسبي است و بر اساس يك پردازندة متوسط موجود در سيستم محاسبه شده است و مستقل از عامل انجام دهندة آن است.

٢- زمان اجرا(Task Completion Time) : اين كميت با توجه به اينكه يك وظيفه توسط كدام عامل (با چه سرعتي) انجام مي‌شود به شكل زير محاسبه مي‌شود:

$$\text{Worst-Case Task-Completion-Time}_{i\ on\ j} = \frac{\text{Average Amount of Work}_i}{\text{Relative Speed}_j}$$

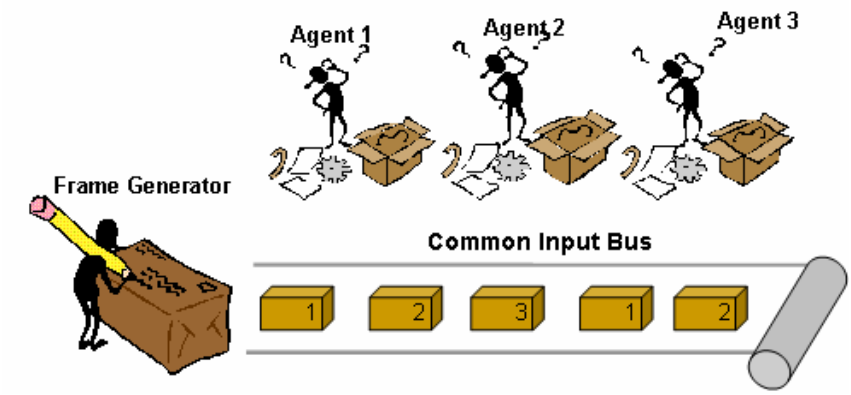
كه فرمول بالا انجام وظيفة i ام توسط عامل j ام را نشان مي دهد. در مورد سرعت نسبي عاملها كمي جلوتر بحث شده است.

٣- مهلت زماني مقرر(Deadline) : اين عدد، بيانگر زماني است كه مهلت زماني مقرر براي انجام يك وظيفه به سر مي‌آيد. اين عدد به نحوي تنظيم شده است كه مهلت زماني مقرر در مورد وظيفه‌هاي مهم‌تر كوتاه‌تر بوده، وظيفه‌هاي كم اهميت‌تر، مهلت زماني طولاني‌تري داشته باشند.

٤- درجة اهميت(Criticality): ميزان اهميت يك وظيفه در مقايسه با ديگر وظايف سيستم توسط اين كميت نسبي نمايش داده مي‌شود.

٥-٤- شرح دقيق هر يك از مولفه‌هاي سيستم

شكل زير يك نماي كلي از مولفه‌هاي موجود در طرح ارائه مي‌كند.



شكل 11. مولفه‌هاي سيستم و نحوة تعامل آنها با يكديگر

مولفه‌هاي اصلي موجود در سيستمي كه به عنوان بستر تحقق ايده‌ها انتخاب شده است، در اينجا با شرح كافي آورده شده اند.

٥-٤-١- Frame Generator

به منظور توليد داده هاي مربوط به هر عامل، اين مولفه بسته‌هاي داده‌اي را توليد مي‌كند و از طريق باس مشترك ورودي در اختيار همة عاملها قرار مي‌دهد. اين مولفه نقطه مهمي در كل سيستم به شمار مي رود. چرا كه توليد داده‌ها يا به عبارت واقعي‌تر جمع‌آوري داده‌هاي مرتبط با هر عامل را بر عهده دارد. از طريق اين مولفه مي‌توانيم با نرخ ثابت يا به صورت متغير تصادفي با توزيع مشخص به توليد داده بپردازيم. در گونه‌اي كه نرخ آمدن داده‌هاي ورودي يكسان و ثابت است، هر عامل بسته‌هاي داده‌اي با پريود ثابت و برابر به عامل منسوب مي‌شوند و عاملها هم از اين نرخ آگاهند و با قطعيت كامل تصميم مي‌گيرند. در مدل غيرقطعي، فركانس توليد داده‌ها براي هر عامل ثابت نيست بلكه از يك توزيع نرمال با ميانگين و واريانس مشخص پيروي مي‌نمايد.

٥-٤-٢- Common Input Bus

به منظور نگهداري خروجي مولفة Frame Generator قبل از اينكه عاملها دادة مورد نظر خود را بردارند اين باس مشترك ورودي در سيستم قرار داده شده است.

Common Help Buses - ۵-۴-۳

به منظور اعلام نياز به كمك و نيز اعلام انجام كمك‌رساني به ديگر كمك‌رسانها، دو باس مشترك، هر يك به صورت FlipFlop هايي به تعداد عاملها در سيستم قرار گرفته است.

Help-Request SRFF Pack - ۵-۴-۴

اين باس كه به شكل مجموعه‌اي از SR-FlipFlop ها پياده‌سازي شده است، توسط هر عامل نيازمند به كمك Set مي‌شود و بدين وسيله به ديگر عاملها اعلام مي نمايد كه دچار خرابي شده و از انجام وظايف خود ناتوان است. در صورت رفع خرابي خود عامل كه قبلا نيازمند كمك بوده، با انجام Reset سلامت خود را به همه اطلاع مي‌دهد.

Is-Helped SRFF Pack - ۵-۴-۵

اين باس نيز به شكل مجموعه اي از SR-FlipFlop ها پياده‌سازي شده است و در صورتي كه يك عامل تصميم به كمك‌رساني به عامل نيازمند به كمك بگيرد،پرچم نظير عامل نيازمند را در روي اين باس چك مي‌كند و در صورتي كه مقدار آن صفر باشد، مقدارش را به يك Set مي‌كند تا از انجام كمك تكراري توسط بقيه جلوگيري شود.پايين آوردن يا (Resetنمودن) اين پرچم به عهدة Frame Generator است تا به محض توليد داده جديدي براي عامل، به ديگران اعلام نمايد كه در مورد اين دادة جديد بايد به يك عامل خاص مجددا كمك شود.

Agents - ۵-۴-۶

در اين سيستم تعدادي عامل مشابه از لحاظ ساختار و متفاوت از لحاظ ويژگي‌هاي دروني وجود دارند كه از تعامل آنها سيستم چندعامله موردنظر ساخته مي‌شود. هرعامل داراي يك ID خاص است كه همين ويژگي، موجب ايجاد تفاوتهايي در هر عامل در مقايسه با ديگر هم تيمي‌هاي وي مي‌شود. اين تفاوتهاي دروني در اينجا شرح داده مي‌شوند:

۱- سرعت اجرايي: اين ويژگي يك عدد نسبي است كه توان محاسباتي پردازندة هر عامل را مشخص مي‌كند و در اصل بيانگر سرعت پردازنده‌اي است كه هر عامل داراست.

۲- قابليت اعتماد: اين ويژگي نيز به صورت يك كميت نسبي ميزان قابليت اعتماد به عملكرد هر عامل يا در اصل پردازندة هر عامل را مشخص مي‌كند.

۳- زمان آزاد: اين كميت، نمايشگر مقدار زماني است كه عامل بعد از انجام وظيفة خودش براي كمك‌رساني در دست دارد. اين عدد تحت تاثير اثري شبيه يادگيري است كه با افزايش اين يادگيري، طول مدت زمان اجراي وظيفة عامل كاهش يافته، زمان آزادي او افزايش مي يابد. به عبارت ديگر هر چه تعداد وظيفه‌هاي تا به‌حال انجام شده توسط هر عامل افزايش يابد، تسلط نسبي عامل بر انجام وظيفه‌هاي مشابه بعدي اضافه مي‌گردد و اين اثر را مي‌توان با كاهشي متناسب با تعداد وظيفه‌هاي تا به‌حال انجام شده در زمان اجراي وظيفه نمايش داد. فرمول زير محاسبة زمان آزادي را در حالت عادي و بدون اثر شبه يادگيري نشان مي‌دهد:

$$\text{Free Time}_i = \text{Period} - \frac{\text{Amount Of Work}_i}{\text{Relative CPU Speed}_i}$$

جدول 1 نحوة تنظيم پارامترها و تقسيم اولية وظيفه‌ها را در مشابه‌سازي‌هاي انجام شده نشان مي‌دهد. البته لازم به توضيح است كه در مورد هر عامل وظيفة حالت عادي آن (و نه وظايفي كه احتمالا در فرآيند كمك‌رساني تقبل خواهد نمود، نشان داده شده است):

جدول 1. نحوة مقداردهي اولية پارامترها و تقسيم اولية وظيفه ها

Agent	Criticality	Deadline	Relative Amount of Work	Relative CPU Speed	Average Own Process Time	CPU Reliability	Fault Probability
1	5	20 NS	12	2	6	Least	Most
2	5	20 NS	6	1	6	Least	Most
3	5	20 NS	6	1	6	Least	Most
4	5	20 NS	12	2	6	Least	Most
5	5	20 NS	18	3	6	Least	Most
6	8	15 NS	6	1	6	Medium	Medium
7	8	15 NS	12	2	6	Medium	Medium
8	8	15 NS	18	3	6	Medium	Medium
9	10	10 NS	6	1	6	Most	Least
10	10	10 NS	12	2	6	Most	Least

تنظيم اعداد بالا، در مورد هر عامل به‌گونه‌اي انجام شده است كه سيستمي با معناي فيزيكي و ملموس حاصل شود، براي مثال:

CPU Speed را عددي صحيح در بازة [1,3] انتخاب نموده‌ايم. بنابراين عاملهايي با سرعت پردازشي ۳ داراي قابليت ويژه فرض مي‌شوند. چرا كه اگر مهلت زماني عاملي در حال از دست رفتن باشد، با احتمال بيشتري به كمك چنين عاملهايي

مي‌توان وظيفة بر زمين مانده را پيش از موعد مقرر به انجام رساند. در توزيع پردازنده‌ها ميان عاملها به نحوي عمل شده است كه كارهايي با بار كاري بيشتر به عاملهايي كه پردازندة سريعتري در اختيار دارند، محول گردد. يعني اگر فرض كنيم كه سه دسته وظيفه با مقادير كاري كم، متوسط و زياد داريم، اين كارها را به ترتيب به عاملهايي با سرعت پردازشي زياد، متوسط و كم آنها محول مي‌نماييم.

قابليت اطمينان عددي صحيح در بازة [1,3] انتخاب شده است. عاملي كه اهميت وظيفة وي بالا است، طبعا بايد پردازندة قابل اعتمادتري داشته‌باشند. بنابراين اين كميت و ضريب اهميت از يكديگر مستقل نيستند.

ضريب اهميت هم عددي نسبي در بازة [1,10] است. به عنوان مثال عامل با ضريب بحران وظيفة ۱۰ داراي بالاترين اولويت خواهد بود.

Fault Generator -۷-۴-۵

اين مولفه، شبيه‌سازي خرابي عاملها را در سيستم بر عهده دارد و با فعال نمودن يك سيگنال BeFaulty فرمان خراب شدن را به هر عامل صادر مي‌كند. البته از آنجا كه احتمال خرابي را مي‌توان با نسبت عكس قابليت‌اعتماد تنظيم نمود و به صورت با معني‌تري خرابي را شبيه‌سازي نمود، اين مولفه توليد خرابي را به صورت متناسب با عكس قابليت اعتماد بر عهده دارد.

۵-۵- فرضيات

در اين سيستم به مسالة تشخيص خطا پرداخته نشده است و فرض بر اين است كه هر عامل قادر است از رخداد خطاي خود آگاهي يابد و اين مساله را با ارسال درخواست كمك به بقيه نيز اطلاع دهد. درخواست كمك در اين سيستم با بالا بردن(Set نمودن) پرچم مربوط به هر عامل به اطلاع بقيه مي‌رسد بنابراين از لحاظ تعداد خرابي‌هاي قابل رخداد در هر لحظه محدوديتي وجود ندارد و تا زماني كه حداقل يك عامل در سيستم فعال باشد سيستم مي‌تواند با وجود كاهش كارايي به عملكرد خود ادامه بدهد.

علاوه بر اين فرض شده است كه يك عامل قادر است وظيفة هم‌تيمي‌هاي خود را نيز در مواقع لزوم انجام بدهد. بدين معنا كه توانايي‌هاي عملياتي عاملها در حد يكديگر است و تفاوتها در مواردي است كه در بالا اشاره شد .

فرض ديگري كه در سيستم لحاظ شده آن است كه ضريب بحران (اهميت) وظيفة هر عامل با عاملهاي ديگر متفاوت است و ميزان اهميت هر وظيفه در كارايي كل سيستم با اهميت ديگري متفاوت است .

مورد ديگر اينكه هر عامل بايد از حجم متوسط وظيفة ديگر عاملها آگاهي داشته باشد تا بتواند در موقع لزوم تخميني از مدت زمان انجام آن وظيفه توسط خودش بدست آورد. اين اطلاع عامل را ياري ميكند تا در مورد امكانسنجي كمك مورد تقاضا تصميم بگيرد.

علاوه بر اينها، زمان واقعي انجام كار هر عامل ثابت نيست بلكه عددي متغير بر اساس دادة ورودي است. بدين ترتيب، زمان پايان وظيفة خود عامل و شروع به كمكرساني در همة تناوبها يكسان نبوده كه اين خود بر پويايي مساله ميافزايد.

نرخ ورود داده ها بر روي باس ورودي كه در اصل به منزلة نرخ محول نمودن وظيفه به عاملها نيز ميباشد، توسط Frame Generator تعيين ميشود كه ميتواند ثابت يا متغير بر اساس يك توزيع نرمال شناخته شده باشد. در فصل ٦ كه طراحيهاي مختلف سيستم بيان خواهد شد به تفصيل به اين مساله خواهيم پرداخت.

٦- روشهاي پيشنهادي

همانطور كه قبلا نيز بيان شد، هدف اين تحقيق ارائة روشهايي است كه در آنها خود عاملها به صورت گسترده و با تغيير نقش، اقدام به كمكرساني به عاملهاي خراب ميكنند تا از كاهش كارايي جلوگيري شده و سيستم با تحملپذيري خرابي بيشتري قادر به ادامة كار باشد. در اين فصل روشهايي كه براي حل اين مساله در شرايط مختلف پيشنهاد شده است، شرح داده ميشوند.

٦-١- مقدمه

در اصل روشهاي مختلف پيشنهادي در اين تحقيق، در طراحي عاملها و مولفة Frame Generator با يكديگر تفاوت دارند و باسهاي مشترك دادههاي ورودي، خروجي و كمك رساني و مولفة شبيهسازي خطا در كلية طراحيها ثابت هستند.

در همة اين روشها، كمكرساني هنگامي آغاز ميشود كه يك عامل با مشكلي مواجه شود. در اين شرايط، بقية عاملها از طريق باس مشترك كمك متوجه ميشوند كه كدام عامل يا عاملها نيازمند كمك هستند و دادة مورد نياز براي انجام وظيفة اين عامل را از باس مشترك داده برميدارند و وظيفة موردنظر را تكميل مينمايند.

آنچه روشهاي مختلف پيشنهادي اين تز را پديد ميآورد، نحوة برخورد عاملها با درخواستهاي كمك است، اعم از اينكه چه موقع به آنها پاسخ بدهند يا اينكه به چه نحوي درخواستهاي كمك را ميان خود تقسيم نمايند يا حتي اينكه به چه ترتيبي عمليات كمكرساني به عاملها را انجام بدهند.

٦-٢- روشهاي مبتني بر تصميمگيري قطعي

٦-٢-١- طرح اول: عاملهايي با تصميمگيري قطعي در مورد امكانسنجي كمك براي تقسيم غيرصريح وظيفهها به صورت مبتني بر سرعت رسيدن به مرحلة كمكرساني

در اين طراحي كه سادهترين طرح براي هدف اين پروژه است، به محض اينكه عامل كمكرسان وظيفة خود را تكميل نمود، شروع به پردازش تقاضاهاي كمك ميكند. اگر بيش از يك عامل نيازمند كمك باشند، عاملهاي كمكرسان بر اساس يك استراتژي كمكرساني توزيع شده كه در مرحلة طراحي تعيين شده است، ترتيب كمكرساني را تعيين ميكنند. سپس عاملها

از ابتداي ليست مرتب بررسي مي‌كنند كه آيا به عامل موردنظر كمك شده است يا خير. اگر اين عامل قبلا توسط عامل ديگري كمك مورد نظر را دريافت كرده‌باشد (كه اين مساله توسط باس مشترك كمك‌رساني با نام IsHelped_SRFF كه در فصل قبل معرفي شد، مشخص مي‌گردد) به سراغ عامل بعدي ليست مي‌روند و به همين ترتيب ادامه مي‌دهند تا اينكه عاملي بيابند كه نيازمند كمك است و بر اساس باس مشتركِ اعلام وضعيت كمك‌رساني، هنوز به وي كمك نشده باشد. در اين لحظه پرچم موردنظر براي اعلام كمك‌رساني به اين عامل را بالا مي‌برند و شروع به انجام كمك مي‌نمايند. پس از تكميل اولين كمك در صورتي كه فرصتي قبل از رسيدن وظيفة بعدي خود عامل باقيمانده باشد و بيش از يك عامل نيازمند كمك باشند، عامل كمك‌رسان مجددا ليست مرتب را بررسي مي‌نمايد و عامل ديگري كه نيازمند به كمك است و هنوز هم كمكي دريافت نكرده است را مي‌يابد و به او كمك مي‌نمايد.

بنابراين چنانكه مشخص است تقسيم درخواست‌هاي كمك ميان عاملهاي كمك‌رسان به صورت ضمني و بر اساس سرعت اجراي وظيفة عامل‌ها صورت مي‌گيرد و هيچگونه هماهنگي مشخصي در اين زمينه ميان عاملها صورت نمي‌گيرد و تنها به كمك مكانيزم شرح داده شده، از ارسال كمك‌هاي تكراري جلوگيري مي‌شود.

٦-٢-١-١- انتخاب مجموعه‌اي از عامل‌هاي نيازمند براي كمك‌رساني

در ابتداي فاز كمك‌رساني، عاملهايي كه زمان كافي براي كمك‌رساني را در اختيار دارند بايد زيرمجموعة مناسبي از عاملهاي نيازمند را برگزينند و كمك موردنظر را به آنها ارائه نمايند. براي اينكه ترتيب كمك‌رساني به اين مجموعه از عامل‌ها توسط كمك‌رسانها تعيين شود، هر كمك‌رسان بر اساس يكي از سياستهايي كه در اينجا شرح داده شده است، عمل مي‌نمايد. اين سياستها به دو دستة ساده و تركيبي تقسيم مي‌شوند.

٦-٢-١-١-١- سياستهاي ساده

استراتژي‌هاي كمك‌رساني ساده اي كه مربوط به ترتيب تخصيص زمان پردازشي عامل به وظايف مورد تقاضا هستند، عبارتند از:

Best Fit (BF) : انتخاب عاملي با طولاني‌ترين وظيفه، به شرطي كه قابل انجام در زمان باقيماندة عامل كمك‌رسان

باشد. در اين سياست، عامل كمك‌رسان زمان لازم براي اجراي وظيفة مورد نياز عامل خراب را از زمان باقيمانده‌اي كه مي‌تواند صرف كمك‌رساني نمايد كم مي‌كند و سپس اين تفاضل را به صورت صعودي مرتب مي‌كند. بدين شكل يك ليست حاصل مي‌شود كه اگر به ترتيب از ابتداي ليست سرويس دهي آغاز شود، سياست BF پياده شده است. با اين سياست، عامل كمك‌رسان طولاني‌ترين وظيفة قابل انجام را برمي‌گزينند با اين فرض كه ديگر عامل‌هايي كه زمان آزاد كمتري دارند، با احتمال بيشتري خواهند توانست ديگر كارهاي كوتاهتر را انجام بدهند.

Shortest Job First (SJF): انتخاب عاملي با كوتاهترين زمان انجام وظيفه، به شرطي كه قابل انجام در زمان باقيماندة عامل كمك‌رسان باشد. اين سياست معمولا توسط عامل‌هاي خودخواه انتخاب مي‌شود چرا كه در اين سياست عامل، كارهاي كوتاهتر را انجام مي‌دهد و كارهاي طولاني‌تر را رها مي‌كند تا عامل‌هاي هم‌تيمي به آنها سرويس‌دهي نمايند.

First Come First Served (FCFS) : انتخاب اولين عاملي كه تقاضاي كمك نموده است، به شرطي كه وظيفة او قابل انجام در زمان باقيماندة عامل كمك‌رسان باشد. در اين سياست به تقاضاهاي كمك به ترتيب، بر اساس زمان رسيدشان سرويس‌دهي مي‌شود.

Earliest Deadline First (EDF) : انتخاب اولين عاملي كه مهلت زماني‌اش به سر خواهد آمد، به شرطي كه وظيفة او قابل انجام در زمان باقيماندة عامل كمك‌رسان باشد.

Most Critical First (MCF) : انتخاب عاملي با مهم‌ترين وظيفه، به شرطي كه وظيفه‌اش قابل انجام در زمان باقيماندة عامل كمك‌رسان باشد. در اين روش، توجه به ضريب بحران يك عامل موجب مي‌شود كه كار عامل‌هاي بحراني‌تر پيش از بقيه انجام گيرد و اين موجب نجات تيم از خرابي كلي مي‌شود.
البته اين دو سياست اخير را كه هر دو اولويت را به مهمترين عامل مي‌دهند (مهم‌ترين عاملها معمولا داراي كوتاهترين Deadline نيز مي‌باشند) مي‌توان با نام First Priority First (FPF) با يكديگر ادغام نمود.

٦-٢-١-١-٢- سياستهاي تركيبي

يك سياست زمانبندي تركيبي در اصل، انجام دو سياست ساده به صورت متوالي است. براي مثال اگر سياست FPF را در نظر بگيريم، ليست حاصل از اعمال اين سياست به تنهايي، ليستي است از عاملهايي كه درخواست كمك صادر كرده اند، مرتب شده بر اساس اولويت اهميت يا مهلت زماني هر وظيفه. در اين ليست، در صورتي كه دو يا چند عامل داراي ضريب بحران برابر يا مهلت زماني برابر باشند، ميتوان از پارامترهاي ديگري چون طول اجراي وظيفة مورد تقاضا يا زمان دريافت تقاضاي كمك استفاده نموده، ليست جديدي بدست آورد كه در آن عاملهاي با ضريب بحران يكسان بر اساس اين اطلاع اضافهتر، مرتب شده اند. اين سياستها را كه روي نتايج اولية FPF عمل ميكنند، به ترتيب SJF_over_FPF و FCFS_over_FPF ميناميم. لازم به توضيح است كه سياست BestFit-over-X يك سياست مناسب براي تعيين ترتيب كمكرساني نيست. چرا كه سياست BestFit براي اختصاص يك وظيفه به يك عامل كاربرد منطقيتري دارد تا در تعيين **ترتيب** اجراي وظيفهها. يعني اينكه اگر بر اساس اين سياست تعيين نماييم كه كدام تقاضاي كمك به كدام عامل منتسب شود (همانطور كه در روش سوم به بعد انجام شده است) بسيار توجيهپذيرتر است تا اينكه يك عامل را مجبور نماييم كه در ابتدا به كارهاي طولانيتر رسيدگي نمايد و سپس كارهاي كوتاهتر را تكميل نمايد. اين سياست چنانكه در [٤٧] هم آمده است، هيچ نتيجهاي به جز طولانيشدن زمان پاسخ سيستم نخواهد داشت.

٦-٢-١-٢- انجام نهايي عمليات كمك

اكنون عامل كمكرسان بر اساس سياستي كه در پيشگرفته است كه FCFS_over_FPF يا SJF_over_FPF است، يك ليست مرتب شده از وظايف براي انجام دادن در اختيار دارد و ميتواند از ابتداي ليست، سرويسدهي را آغاز نمايد تا اينكه زمان باقيمانده اش به پايان برسد. وقتي زمان باقيمانده به پايان برسد، m عامل (نه لزوما كلية عاملهاي نيازمند كمك)، كمكرساني شده اند. عاملهايي كه به آنها كمك نشده است، ممكن است در همين فاز توسط ديگر عاملها كمكرساني شوند. البته ممكن است، همين عامل كمكرسان كه فعلا قادر به كمك نبوده در گامهاي بعدي به اين عاملها كمك نمايد.

٦-٢-٢- طرح دوم براي بهبود طرح اول : سيستم اول به همراه اِعمال ملاحظاتي براي انتخاب كمك‌رسان بهتر

در اين طرح نيز همانند طرح اول، تنها مساله‌اي كه موجب رقابت ميان عامل‌هاي كمك‌رسان مي‌شود، سرعت رسيدن عامل‌ها به مرحلة كمك‌رساني است. امّا آنچه اين طرح را با طرح اول متفاوت مي‌سازد، ملاحظات بيشتري است كه عامل‌هاي كمك‌رسان پيش از اقدام به كمك در نظر مي‌گيرند.

عاملهاي كمك‌رسان در هنگام مواجهه با يك درخواست كمك (كه به ترتيب از ابتداي ليست مرتب برمي‌دارند)، پارامترهاي زير را براي تشخيص مناسب بودن عمليات كمك رساني خود مورد بررسي قرار مي‌دهند:

١) قابليت اطمينان خود
٢) درجه اهميت وظيفة عامل نيازمند كمك
٣) توجه به داشتن قابليت ويژة خود و اينكه آيا كمك‌رساني نيازمند استفاده از قابليت ويژه است يا خير؟

در صورتي كه درجة اهميت كار عامل نيازمند كمك زياد باشد و قابليت اعتماد عامل كمك‌رسان چندان قابل‌توجه نباشد، عمليات كمك‌رساني نياز به تامل بيشتر دارد.

در اين شرايط عامل كمك‌رسان بررسي مي‌كند كه آيا كمك‌رسان مناسبتري از لحاظ قابليت اعتماد به پردازنده‌اش وجود دارد يا خير. نحوة انجام اين بررسي چنين است كه هر عامل با توجه به باس درخواستهاي كمك قادر است تعيين نمايد كه (كدام عاملها در حال حاضر نيازمند كمك هستند) كدام عاملها در حال‌حاضر قادر به كمك‌رساني هستند. با توجه به شناختي كه هر عامل از ديگر هم‌تيمي‌هايش دارد (از جمله قابليت اعتماد به عملكرد هر كدام را مي‌داند)، قادر است كه تعيين نمايد كه آيا كمك‌رسان مناسبتري از اين نظر در سيستم به صورت فعال وجود دارد يا خير. اگر چنين باشد، از اقدام به كمك صرفنظر مي‌كند و اين فرصت را در اختيار عامل‌هاي مناسبتر قرار مي‌دهد.

به همين‌ترتيب اگر مهلت زماني عامل نيازمند كمك در حال سپري شدن باشد و عامل كمك‌رسان آنقدر سريع نباشد كه بتواند وظيفه را پيش از موعدمقرر تكميل نمايد، اين عامل، در صورت اقدام به كمك، فرصت كمك‌رساني را از يك عامل مناسبتر كه به دليل داشتن قابليت ويژه (سرعت بالاي

محاسباتي) مي‌توانسته كمك‌رسان بهتري باشد، از او گرفته است. بنابراين در چنين حالتي نيز بهتر است كه عاملِ اول، اقدام به كمك‌رساني ننمايد و كمك‌رساني را به ديگري واگذار نمايد.

همانطور كه ذكر شد ، اين طرح تفاوت عمده‌اي با طرح اول ندارد و از لحاظ عدم تقسيم صريح درخواست‌هاي كمك شبيه طرح اول است.

٦-٢-٣- طرح سوم براي بهبود طرح اول : سيستمي با تصميم‌گيري قطعي و مبتني بر الگوريتم صريح نسبتا بهينه براي تقسيم وظيفه‌هاي نيازمند كمك ميان عاملهاي كمك‌رسان

در دو طرح قبلي، تقسيم وظايف نيازمند كمك بر اساس سرعت رسيدن عاملها به مرحله تصميم‌گيري و اقدام به كمك صورت مي گرفت، يعني كليه عاملها يك ليست مرتب (بر اساس سياست واحد و مشخصي) از وظيفه ها را كه براي كليه كمك رسانها مشابه هم محاسبه مي‌شود، در پيش رو دارند. آنچه موجب مي شود كه يك كمك رسان به يك كمك خواه متفاوت با آنكه كمك رسان هم تيمي وي انتخاب كرده است، سرويس دهي نمايد، سرعت انتخاب عاملها از ليست مرتب است كه با اتخاذ روشهاي جلوگيري از Conflict به شيوه سخت افزاري در نهايت منجر به ايجاد عملكرد صحيح و قابل قبول مي‌گردد. به بيان خلاصه در دو طرح اول، تاكيدي بر نحوه تقسيم وظيفه ميان عاملها نبوده، بلكه بيشترِ توجه به اينكه" ترتيب كمك رساني عاملها بر چه اساسي باشد" معطوف بوده است و تا كنون الگوريتم تقسيم وظيفه ميان عاملها *نه به صورت صريح بلكه به صورت ضمني* بر اين اساس بوده است : "*هر عامل كه به مرحله كمك رساني رسيد از ابتداي صفِ بالاترين اولويت ها، يك وظيفه را بر دارد و در صورتي كه اين كمك قبلا توسط نفر ديگري انجام نشده باشد، آن را انجام دهد، در غير اينصورت وظيفه با اولويت بعدي را بررسي نمايد.*"

اما در روش سوم مبناي كار اين است كه: انتخاب بهترين و مناسب‌ترين كمك‌خواه ها توسط كمك‌رسان‌هاي واجد شرايط صورت گيرد و به عبارت دقيق‌تر، وظيفه‌هاي بر زمين‌مانده ميان عاملهاي كمك‌رسان صريحا تقسيم گردد و به الگوريتم تقسيم ضمني كه در بالا بيان شد، اكتفا نشود.

در راستاي نيل به اين هدف، بررسي الگوريتمهايي براي تقسيم وظيفه و زمانبندي در حوزة سيستمهاي چندپردازنده اي بلادرنگ كه در فصل چهارم اصول كلي آنها بيان شد، نشان مي‌دهند كه اين روشها :

غالبا ذات متمركز داشته، توسط يك مولفة آگاه از اطلاعات ديگر اجزاء سيستم، به اجرا در مي آيند.

در اين روشها كار تا پايان اتخاذ تصميم متوقف مي‌ماند و بعد از مشخص شدن كارهايي كه هر ماشين بايد انجام بدهد، كليه واحدها با هم شروع به پردازش مي كنند.

اين الگوريتم هاي بهينه غالبا از پيچيدگي‌زماني NP-Complete برخوردارند و در اين حوزه يك زمينة تحقيقاتي مهم، ارائة الگوريتم هاي نسبتابهينه با پيچيدگي كمتر است. منظور از پيچيدگي كمتر، حداقل درجة چندجمله اي است.

غالب اين روشها ابتكاري[1] بوده، محدوديت هاي فيزيكي يا زماني براي پردازنده ها يا وظيفه ها قائل نيستند كه اين خود يكي از معايب استفاده از چنين الگوريتمهايي در سيستمي شبيه سيستم اين پروژه مي‌باشد، چرا كه در سيستم ما برخي وظيفه ها (كمك‌ها) تنها قابل محول نمودن به برخي عاملها بوده ـ محدوديت فيزيكي ـ و نيز برخي عاملها از لحاظ زماني امكان كمك به برخي عاملهاي ديگر را ندارند ـ محدوديت زماني ـ .[۳۵] ، [۳٦] و [۳۷] .

بنابراين الگوريتمي كه در اين طرح براي تقسيم وظيفه ميان عامل‌ها ارائه شده است ، داراي مزاياي زير است:

۱) به سادگي قابليت توزيع شدگي را دارد كه در همين فصل به آن اشاره شده است.
۲) از پيچيدگي زماني كمتر از $O(n^2)$ برخوردار است.
۳) پياده سازي آن به ويژه در سيستم هاي واقعي سخت‌افزاري ساده است.
٤) محدوديتهاي مورد نياز در طرح (فيزيكي/ زماني) را در نظر مي گيرد.
٥) چون بر مبناي تصميم‌گيري منطقي[2] است بر الگوريتمهاي جستجوي حريصانه[3] برتري دارد.
٦) بار بسيار زياد ناشي از Coordination / Communication موجود در الگوريتمهاي بهينه را ندارد.

البته لازم به توضيح است كه در روند عملكرد سيستم فعلي يا هر سيستم موازي نوعي، همه عاملها در آن واحد

1 Heuristic
2 Rational Decision-Making
3 Greedy Search

به مرحله تصميم‌گيري در مورد كمك نمي‌رسند، برخي عاملها سريع‌تر و برخي ديرتر وارد اين فاز مي‌شوند. به همين دليل است كه نبايد انتظار داشت در اجراي اين الگوريتم يا هر الگوريتم گستردة ديگر، عامل‌ها همه در يك لحظه اقدام به تصميم‌گيري دسته جمعي و احيانا اطلاع‌رساني در برخي موارد به يكديگر نمايند.

در اينجا ابتدا الگوريتم بهينه‌اي كه براي اين هدف مي‌توان پيشنهاد نمود ولي از لحاظ درجة پيچيدگي و پياده‌سازي گسترده مشكل‌ساز است، ارائه مي‌شود و سپس به بيان الگوريتم نسبتا بهينه‌اي كه در بالا مزاياي آن شرح داده‌شد، مي‌پردازيم:

٦-٢-٣-١- الگوريتم بهينة تقسيم وظايف مبتني بر جستجوي كامل از درجه پيچيدگي NP-Complete براي تخصيص m وظيفه به n پردازنده

١- براي هر پردازنده، كليه تركيبات m تايي به پايين وظيفه‌ها را كه از لحاظ زماني امكان‌پذير هستند، تعيين مي‌كنيم.

٢- اگر وظيفه‌اي تنها در ليست تركيب‌هاي يك پردازنده آمده است آن تركيب را نهايي كرده ، آن را از ليست كليه پردازنده هاي ديگر حذف كن.

٣- اگر تركيبي با ماكزيمم تعداد عناصر توسط يك پردازنده امكان‌پذير است، آن تركيب را نهايي كرده، عناصر آن تركيب را از ليست همة ديگر پردازنده ها حذف كن.

٤- اگر تركيباتي با تعداد مساوي از وظيفه ها روي تعدادي پردازنده امكان‌پذيرند، پردازنده را بر اساس شرط Best Fit انتخاب كن.

٥- وظيفه‌هاي باقيمانده را به ليست نهايي منتقل كن تا به ترتيب اجرا گردند.

درجة پيچيدگي اين الگوريتم به دليل مبتني بودن بر جستجوي كامل n وظيفه در شبكه‌اي از mپردازنده از درجة $O(n^m)$ است.

٦-٢-٣-٢- الگوريتم بهينة مركزي براي تقسيم وظيفه‌ها ميان عاملها يا زمان‌بندي در شرايط بروز خطا[1]

١- درخواست‌ها را بر اساس درجه اهميت / مهلت زماني و سپس بر اساس طول انجام كارشان مرتب كن.

٢- كمك‌رسان‌ها را براساس قابليت اطمينان به عملكرد آنها و سپس بر اساس زمان آزادشان (Initial Free Time) مرتب كن:

Sort by Reliability;

Sort by Free Time;

$$\text{Free Time}_i = \text{Period} - \frac{\text{Amount Of Work}_i}{\text{Relative CPU Speed}_i}$$

اين زمان آزاد است كه به صورت افزونگي ضمني براي چنين مواقعي در سيستم به صورت رزرو باقي مانده است و اكنون بايدصرف كمك‌رساني بشود.

٣- اگر عاملي داراي قابليت‌ويژه‌اي است، فعلا آن را از ليست كمك‌رسانها حذف كن تا براي مواقع ضروري‌تر آزاد بماند. (اگر زمانبندي بدون نياز به او شدني باشد بهتر است كه اين عامل بي كار بماند.)

٤- مهم‌ترين درخواست كمك را به قابل اعتمادترين عامل كه وقت آزاد كافي هم دارد محول كن، (در صورتي كه چند انتخاب داري بر اساس شرط BestFit عمل كن) زمان باقيمانده پس از اختصاص را محاسبه كن و در ليستي جديد با نام Current Free Time بنويس. در مورد عاملي كه يك بار وظيفه‌اي به او ارجاع شده به ليست وقت آزادش توجه كن و از او در زمانبندي‌هاي مجدد استفاده كن. (از بين دو عامل كه هر دو به يك اندازه قابل اعتمادند و زمان هر دو كافي است عاملي كه كار موردنظر را باسرعت بيشتري انجام مي‌دهد، انتخاب بهتري است). اين مرحله را آنقدر تكرار كن تا يا كمك‌رسانها و يا كمك‌خواه‌ها تمام شوند.

علت انتساب وظيفه‌ها بر اساس BestFit (كه در شرح روش اول به آن اشاره شد) آن است كه بكارگيري اين سياست قادر است، زمان آزاد هرز عامل‌هاي كمك‌رسان را به مي‌نيمم برساند. يعني عاملي كه انجام يك وظيفه با زمان اجراي معلومي براي او زمان هرز و تلف شدة كمتري باقي مي‌گذارد، انتخاب بهتري است تا عاملي كه زمان تلف شدة او بيشتر است. به عبارتي، مشابه اختصاص حافظه به فرآيندها براي اجرا

1 An Optimal Central Task Distribution / Scheduling Strategy in Fault Situation

در سيستمهاي عامل، مينيمم‌سازي زمان تلف‌شده مزيتي است كه سياست BestFit به‌دست مي‌دهد.

۵- اگر وظيفه‌اي وجود دارد كه هنوز كمك‌رساني براي او پيدا نشده و با شرايط فعلي زمان كافي براي انجام اين وظيفه توسط هيچ كمك‌رساني وجود ندارد، زمانبندي‌هاي انجام شده را ناديده بگير، عامل با قابليت ويژه (سرعت زياد) را (كه قبلا در زمانبندي از آن استفاده نشده بود) به ليست عاملهاي آماده براي كمك‌رساني اضافه كن (در شرايط فعلي وظيفه‌هاي غير قابل انجام را در UnSchedulable1 ذخيره كن) و برو به ٤.

٦- اگر وظيفه‌اي وجود دارد كه هنوز كمك‌رساني براي او پيدا نشده و با شرايط فعلي زمان كافي براي انجام اين وظيفه توسط هيچ كمك‌رساني وجود ندارد، زمانبندي‌هاي انجام شده را ناديده بگير، عامل با قابليت ويژه (سرعت زياد) را (كه قبلا در زمانبندي از آن استفاده نشده بود) به ليست عاملهاي آماده براي كمك‌رساني اضافه كن، عامل‌هاي كمك خواه را بر اساس زمان مورد نيازشان و نه بر اساس درجه اهميت كارشان مرتب كن (در شرايط فعلي وظيفه‌هاي غير قابل انجام را در UnSchedulable2 ذخيره كن) و برو به ٤.

٧- اگر كليه عاملها پوشش داده شدند، كار زمانبندي به پايان رسيده است و الا بايد ميان دو زمانبندي كه تا به حال ارائه شده بهترين از لحاظ كارايي مشخص شود. اگر $\sum_{i=1}^{NoofUnSchedulable1} C_i$ كمتر از $\sum_{i=1}^{NoofUnSchedulable2} C_i$ باشد زمانبندي نهايي را بر اساس نتيجه ۵ قرار بده و الا زمانبندي نهايي نتيجه ٦ خواهد بود. اگر از اين لحاظ مانند هم هستند روشي كه عاملهاي كمتري را درگير مي‌كند برتري دارد.

٨- اكنون هنگام انجام واقعي وظيفه‌هاست: اگر به يك عامل بيش از يك وظيفه ارجاع شده است، بر اساس SJF به ليست كارهايش رسيدگي كند.

اكنون به نحوة گسترده نمودن اين الگوريتم در ميان عامل‌ها مي‌پردازيم:

در اين الگوريتم هر عامل لازم است كه از زمان آزاد (Free Time) همة عاملهاي ديگر و نيز زماني كه هر وظيفه توسط يك عامل ديگر اجرا مي‌شود، مطلع باشد. اين دو نياز اطلاعاتي به سادگي و بدون نياز به اعلام در زمان اجرا (كه مي‌تواند نيازمند ارتباطات صريح باشد) مي‌تواند برآورده شود. به اين شكل:

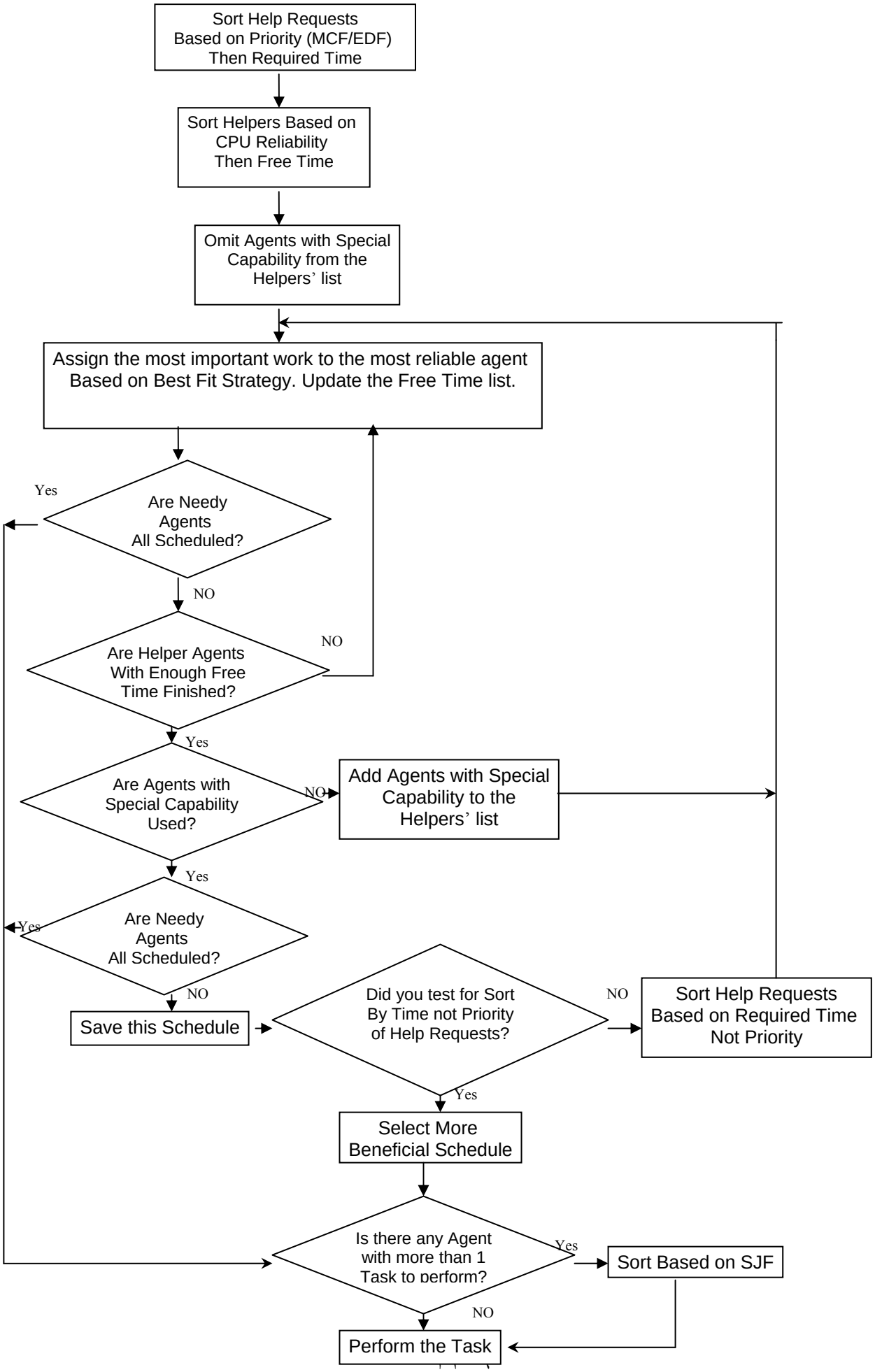
هر عامل از سرعت نسبي ديگر عاملها اطلاع داشته باشد.
هر عامل از حجم متوسط تمام وظيفه هاي سيستم (مستقل از اينكه انجام دهندة آن چه كسي است) اطلاع داشته باشد
Period (فعلا ثابت و يكسان) رسيدن وظيفه هاي جديد را بداند.

بنابراين مي‌تواند دو اطلاع زماني زير را محاسبه كند:

$$\text{Worst-Case Task-Completion-Time}_{i\,on\,j} = \frac{\text{Amount Of Work}_i}{\text{Relative CPU Speed}_j}$$

$$\text{Free Time}_i = \text{Period} - \frac{\text{Amount Of Work}_i}{\text{Relative CPU Speed}_i}$$

البته از آنجا كه سعي شده طول زمان انجام وظيفة يك عامل ثابت نباشد (به ۲ طريق: ۱- استفاده از يك Switch case به منظور متغير نمودن اين زمان در يك Range مشخص و ۲- اعمال اثر يادگيري در سرعت اجراهاي بعدي) ، اين محاسبه داراي واريانسي خواهد بود كه اگر اين واريانس زماني از كوتاهترين وظيفه قابل انجام كوتاهتر باشد، مي‌توان اثر آن را ناديده گرفت.
به منظور خلاصه نمودن ايدة اصلي الگوريتم نسبتا بهينة تقسيم وظيفه، جايگاه هر يك از سياستهاي ساده زمانبندي در اين الگوريتم در اينجا نشان داده‌شده است:
براي تعيين ترتيب محول نمودن وظيفه ها به عاملها بر اساس MCF يا EDFعمل مي‌شود.
براي اينكه بدانيم وظيفه هاي مرتب شده در بالا به كدام كمك‌رسان محول شوند از BF استفاده مي‌كنيم.
هر عامل كمك‌رسان در صورتي كه چند وظيفه براي انجام دادن داشته باشد، مي‌تواند بر اساس SJF يا FCFSعمل نمايد.
البته با توجه به مطالعات گسترده اي كه در حوزة زمانبندي در سيستم هاي عامل انجام شده است و كاربرد آنها در سيستم هاي گسترده [۴۷] ، SJF زمان پاسخ بهتري به دست مي‌دهد
عملكرد اين سيستم مطابق فلوچارتي است كه در صفحة بعد آورده شده است.


Sort Help Requests Based on Priority (MCF/EDF) Then Required Time
Sort Helpers Based on CPU Reliability Then Free Time
Omit Agents with Special Capability from the Helpers' list
Assign the most important work to the most reliable agent Based on Best Fit Strategy. Update the Free Time list.
Are Needy Agents All Scheduled?
Yes
NO
Are Helper Agents With Enough Free Time Finished?
NO
Yes
Are Agents with Special Capability Used?
NO
Add Agents with Special Capability to the Helpers' list
Yes
Are Needy Agents All Scheduled?
Yes
NO
Save this Schedule
Did you test for Sort By Time not Priority of Help Requests?
NO
Sort Help Requests Based on Required Time Not Priority
Yes
Select More Beneficial Schedule
Is there any Agent with more than 1 Task to perform?
Yes
Sort Based on SJF
NO
Perform the Task

٦-٣- روشهاي مبتني بر تصميم‌گيري غيرقطعي

٦-٣-١-طرح چهارم به عنوان تعميم طرح سوم : سيستم با تصميم‌گيري غيرقطعي از لحاظ زمان رسيدن وظيفه بعدي خود عامل كمك‌رسان

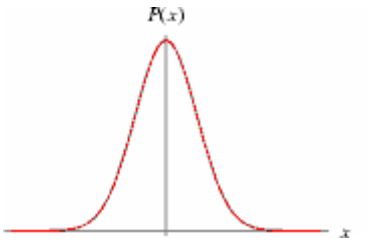
اين طرح به عنوان تعميمي براي سيستم قبل (روش سوم) توسعه داده شده است. در اين طرح، عامل‌هاي كمك‌رسان پس از تقسيم وظايف نيازمند كمك‌رساني، با تصميم‌گيري غيرقطعي به بررسي امكان‌سنجي انجام‌پذير بودن وظيفة مورد تقاضا براي كمك‌رساني از طريق محاسبة احتمالاتي زمان رسيدن وظيفة بعدي خود مي‌كنند و در صورتي كه با احتمال مناسبي اين كار را امكان‌پذير بيابند، اقدام به انجام كمك مي‌نمايند و در غير اينصورت از انجام آن منصرف مي‌شوند.

در اين طرح، مولفة Frame Generator كه تا به حال با پريود ثابت و قطعي اقدام به توليد وظيفه مي‌نمود، وظيفه‌هايي با پريودي كه از توزيع احتمالاتي برخوردار هستند، براي عاملها توليد مي‌كند. البته عاملها از ميانگين و انحراف معيار توزيع مطلعند و به همين وسيله قادرند كه با عدم قطعيت اقدام به تصميم‌گيري نمايند.

٦-٣-١-١- ايجاد عدم قطعيت در سيستم

مهم‌ترين پارامتري كه در اين سيستم بايد به صورت غيرقطعي تبديل شود تا بر جامعيت طرح افزوده گردد، پريود رسيدن وظيفه براي هر عامل است. به عبارت ديگر، بهتر است كه به جاي يك باس سنكرون يك باس آسنكرون داشته باشيم. بنابراين يك توزيع گوسي با توزيع مانند فرمول(١) و مطابق شكل زير جايگزين عدد پريود ثابتي كه در ويرايش هاي ديگر سيستم موجود بود، گرديد:

$$\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x} e^{-(x'-\mu)^2/(2\sigma^2)}\,dx'. \qquad (١)$$



شكل 12 . تابع توزيع گوسي نمونه، مورد استفاده در Frame Generator

در چنين شرايطي Frame Generator با توزيعي مطابق شكل بالا ولي همچنان به صورت نوبتي براي هر عامل داده ارسال مي كند. نوبتي بودن ارسال يك نكته اساسي است، چرا كه در صورت نوبتي نبودن (تصادفي بودن) ترتيب ارسال داده براي عاملها، نمي توان دانشي در اختيار عاملها قرار داد تا بر اساس آن تصميمي براي كمك بگيرند. بنابراين هر عامل پس از ارسال دادة عاملهاي ديگر مجددا انتظار دريافت يك داده جديد دارد.

در اينصورت به عنوان مثال در صورت وجود ۱۰ عامل در سيستم، پس از گذشت زماني برابر مجموع پريودهاي غيرقطعي ۱۰ عامل دوباره نوبت به ارسال داده يك عامل مي‌رسد. در نتيجه از آنجا كه مجموع N عدد تصادفي با توزيع گوسي با ميانگين m و انحراف معيار s ، خود يك توزيع گوسي با ميانگين mN و انحراف معيار $\sqrt{Ns^2}$ است ، توزيعي كه هر عامل در نظر مي‌گيرد تا بر اساس آن تصميم بگيرد كه آيا به عامل نيازمند كمك با زمان مشخص پردازشي (روي CPU ي خود آن عامل) كمك بكند يا خير كاملا مشخص خواهد بود.

در اينصورت هر عامل بايد محاسبه نمايد كه با چه احتمالي كار كمك‌رساني به يك عامل نيازمند پيش از رسيدن كار بعدي خودش به پايان خواهد رسيد. بنابراين محاسبة احتمال بالا، به اين شكل خواهد بود:

$$X = NextTaskAssignment$$

$$T = Now + CompletionTime_{HelpTask} + WorstCase\_CompletionTime_{NextMyOwnTask}$$

$$P(x > T) = 1 - P(x < T) = \int_{-\infty}^{T} p(x).dx$$

كه عبارت $P(x<T)$ همان مقدار تابع توزيع تجمعي گوسي در نقطة T است و طبق فرمول زير محاسبه مي‌شود:

$$\frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right],$$

كه در آن erf همان تابع معروف است (Error Function) كه در اينجا آورده شده است:

$$erf(x) = \frac{2}{\sqrt{\pi}}.\int_{0}^{x} e^{-t^2}$$

با توجه به اين مطالب، آنچه هر عامل پيش از كمك رساني بايد محاسبه نمايد به اين شكل خلاصه مي‌شود:

$$P(x>T)=1-P(x<T)=1-\frac{1}{2}[1+erf(\frac{x-\mu}{\sigma\sqrt{2}})]$$

در صورتي كه اين احتمال از حد قابل قبولي كه آن را *احتمال ريسك* ميناميم بيشتر باشد، عامل ميتواند با دلگرمي نسبي به اينكه در صورت اقدام به كمك، خللي در انجام وظيفة بعدي وي رخ نميدهد، شروع به كمك كند.

احتمال ريسك ميتواند در چند سطح مختلف تغيير نمايد تا بهترين وضعيت از لحاظ سيستم تعيين شود. اين سطوح مختلف عبارتند از:

١) احتمال ريسك زياد
٢) احتمال ريسك متوسط
٣) احتمال ريسك كم
٤) احتمال ريسك تطبيقي

اين سطوح در فصل آزمايشها و نتايج به صورت دقيقتري معرفي شده اند.

البته بديهي است كه به دليل احتمالاتي بودن شرايط و عدم قطعيت در تصميمگيري، در چند درصد موارد، ممكن است كه عامل وظيفة خود را از دست بدهد.

براي رفع اين مشكل ٢ راه حل وجود دارد :

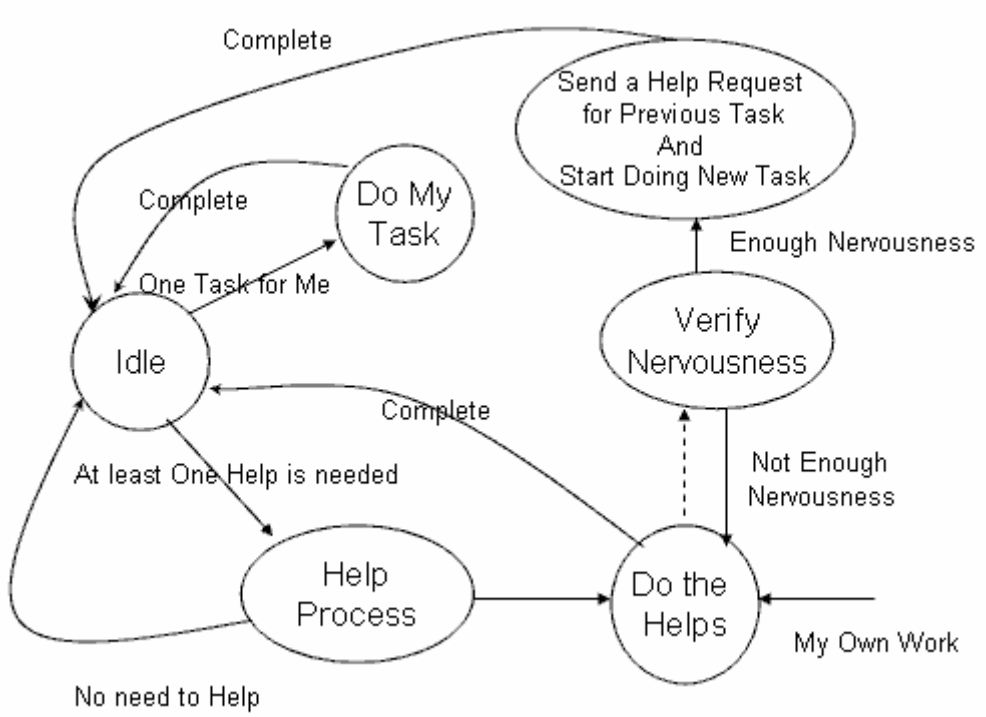
١- عامل كمكرسان در صورت مشغول بودن به كمك براي وظيفة خود درخواست كمك صادر كند كه اين روش در همين طرح به وسيلة عاملها به اجرا درميآيد.

٢- در صورت رسيدن وظيفة خود عامل، در صورت مشغول بودن وي به كمكرساني، به عامل كمكرسان وقفه داده شود تا او پس از طي يك فاز اضافة تصميمگيري، به اين نتيجه برسد كه آيا اتمام وظيفة كمكرساني مهمتر است يا پرداختن به وظيفة خودش. اين راه حل نيز در طرح پيشنهادي آخر توسط عاملها به اجرا درميآيد.

٦-٣-٢- طرح پنجم براي تكميل طرح چهارم: سيستم مبتني بر وقفه‌دهي درصورت رسيدن وظيفة خود عامل و تصميم‌گيري بر اساس محاسبة اضطراب [1]

در اين طرح كه به عنوان طرح تكميلي روش چهارم مطرح است، باز هم عاملها با تصميم‌گيري غيز قطعي مواجه هستند. اما در اينجا، عامل كمك‌رسان در صورت دريافت وقفه‌اي از سوي سيستم به دليل وجود وظيفه‌اي منتظر براي خودش، ابتدا وارد فاز بررسي ميزان اضطراب خود در انجام وظيفة خود مي‌شود. بدين معنا كه ميزان تمايل خود را بر ادامة كار كمك‌رساني يا نگراني پرداختن به وظيفة جديدي كه براي خودش رسيده است، محاسبه مي‌نمايد. در صورتي كه به ادامة كار تمايل بيشتري داشته باشد، براي انجام وظيفة خود درخواست كمك صادر مي‌كند و به ادامة كارقبلي خود مي‌پردازد (مانند روش چهارم) و اگر نگران پرداختن به وظيفة خودش باشد، كار كمك‌رساني را با وجود اينكه تا حدي انجام داده است، رها مي‌نمايد و پرچم مربوط به اعلام وضعيت كمك به آن عامل را كه خود بالا برده بود، مجددا Reset مي‌نمايد تا امكان كمك را به ديگران منتقل نمايد. عملكرد دقيق اين سيستم در دياگرام حالت زير نشان داده شده است:

[1] Nervousness



شكل 13. دياگرام حالت سيستم در طراحي مبتني بر وقفه‌دهي و محاسبة ميزان اضطراب

٦-٣-٢-١-نحوة محاسبة ميزان اضطراب در عاملها در لحظة دريافت وقفه

در محاسبة ميزان اضطراب عامل كمك‌رسان در لحظة دريافت وقفه،عوامل زير دخيل هستند:

١- ميزان باقيمانده از كار فعلي عامل كمك‌رسان به صورت مستقيم:

اگر عامل مقدار قابل توجهي از كار فعلي خود را انجام داده باشد،به احتمال قوي تمايل چنداني به سوئيچ نمودن به يك كار جديد ندارد و شايد عاقلانه‌تر هم اين باشد كه در چنين شرايطي عامل به كار جديد سوئيچ ننمايد.

٢- زمان باقيمانده به از دست رفتن مهلت زماني كار فعلي عامل كمك‌رسان به صورت مستقيم:

در صورتي كه زمان اندكي به از دست رفتن مهلت زماني كار فعلي عامل باقيمانده است، به احتمال زياد تكميل اين وظيفه در صورت توقف و سپس محول شدن به عامل ديگر، كار چندان عاقلانه‌اي نيست و بهتر است كه همين عامل كه مقداري از اين كار را هم تكميل نموده است، اين كار را تكميل نمايد.

٣- زمان باقيمانده به از دست رفتن مهلت زماني كار جديد عامل كمك‌رسان به صورت معكوس.

از طرف ديگر، اگر مهلت زماني كار جديد نيز رو به اتمام است، عامل كمك‌رسان بايد به اين مساله توجه نموده و به مسالة كمك‌رساني رغبت نشان بدهد.

٤- درجة اهميت كار فعلي به صورت معكوس

در صورتي كه اهميت كار فعلي عامل براي سيستم زياد باشد، عامل بايد رغبت كمتري به رها نمودن اينكار و پرداختن به وظيفة جديد از خود نشان دهد.

٥- درجة اهميت كار جديد به صورت مستقيم

در صورتي كه اهميت كار جديدي كه از عامل براي انجام آن مدد خواسته‌اند، زياد باشد، عامل بايد با تمايل بيشتري به رها نمودن كار فعلي خود و پرداختن به وظيفة جديد اقدام نمايد.

٦- ميزان اين احتمال كه وظيفة خود عامل پس از تكميل كار فعلي بيايد، به صورت معكوس.

عاملي كه قرار است از ميان دو كار يكي را برگزيند، بايد در مورد نفع آتي خود نيز بينديشد. بديهي است كه عامل نگران كاري باشد كه با احتمال بيشتري پيش از رسيدن وظيفة جديد خودش به اتمام رسيده باشد.

٧- ميزان اين احتمال كه وظيفة خود عامل پس از تكميل كار جديد بيايد، به صورت مستقيم.

به همين ترتيب يك عامل كمك‌رسان، كار جديد را در صورتي كه با احتمال زيادي پيش از تكميل كار جديد خودش به پايان برسد بيشتر ترجيح مي‌دهد.

٨- مسالة نياز يا عدم نياز كار فعلي و كار جديد به قابليت ويژة عامل در صورتي كه عامل داراي قابليت ويژه باشد، كه بر كل نتيجة پارامترهاي بالا غالب است.

در مورد مسالة قابليت ويژة يك عامل، توجه به اين مساله مهم است كه اگر عامل داراي قابليت ويژه (كه در اين سيستم فعلا به سرعت پردازشي اتلاق مي‌شود)، باشد :

عامل بايد توجه كند كه كدام‌يك از دو وظيفة فعلي يا جديد نيازمند اين قابليت هستند. طبيعي است كه اگر تنها يكي از اين دو وظيفه نيازمند قابليت ويژة عامل باشد، برتري با آن وظيفه است، خواه اين وظيفه، وظيفة جاري باشد يا اينكه جديدا به عامل ارجاع شده باشد. براي تشخيص اينكه كدام وظيفه، نيازمند قابليت ويژة عامل كمك‌رسان است از حجم نسبي وظيفه‌ها كه اطلاعي است كه هر كمك‌رسان در مورد ديگر عاملها دارد، استفاده مي‌كنيم. بدين ترتيب اگر يكي از دو وظيفه، نيازمند قابليت ويژه

باشد كه رغبت عامل به انجام آن وظيفه، بسيار بالا مي‌رود تا حدي كه ديگر پارامترهايي كه در بالا شرح داده شد را تحت تاثير قرار مي‌دهد.

امّا اگر عاملي داراي قابليت ويژه نباشد يا اينكه اگر قابليت ويژه‌اي دارد هيچ يك از دو عامل ديگر نياز به اين قابليت نداشته باشند و بالاخره اگر هر دو عامل نيازمند اين قابليت باشند، بدون در نظر گرفتن پارامتر قابليت ويژه و تنها با احتساب ساير مواردي كه در بالا بيان شد، عامل كمك‌رسان اقدام به انتخاب يك وظيفه مي‌نمايد.

اكنون پس از شرح هر پارامتر به بيان فرمول كلي محاسبة اضطراب مي‌پردازيم:

$$Nervousness = \begin{cases} 0, \text{ if CurrentTask needs my Special Capability} \\ 1, \text{ if New Task needs my Special Capability} \\ k_1 . \mathrm{Re}maining Work_{Current} . k_2 . \dfrac{Time2MissDeadline_{Current}}{Time2MissDeadline_{New}} \\ . k_4 . \dfrac{P(NewTask\_CompletesBefor\_MyNewTask)}{P(CurrentTask\_CompletesBefor\_MyNewTask)} . k_3 . \dfrac{Criticality_{New}}{Criticality_{Current}}, \text{ otherwise} \end{cases}$$

كه در فرمول بالا $k1$، k_2، $k3$ و $k4$ وزن متناظر با اثر هر پارامتر در محاسبة اضطراب است كه در ساده‌ترين حالت و به منظور تاثيرگذاري يكسان، مي‌توان همه را برابر يك فرض نمود.

بنابراين به طور خلاصه:

اين روش نيز مانند روش قبل، مبتني بر تصميم‌گيري احتمالاتي است و تنها تفاوت آن با روش قبل در نحوة پرداختن به انجام وظيفة خود عاملهاي كمك‌رسان است. در اينجا بر خلاف روش قبلي، كار كمك‌رساني جاري عاملها، با رسيدن وظيفه‌اي براي خودشان دچار وقفه مي‌شود. پس از بررسي ميزان اضطراب، در صورتي كه اين عدد از يك آستانه (كه در فصل بعدي با آزمايش مقدار مناسب آن تعيين شده است) بيشتر باشد، عامل كار جاري خود را رها نموده، به سراغ انجام وظيفة خودش مي‌رود و در صورت عدم تمايل به اين

تغيير وظيفه (به دليل مضطرب نبودن)، به تكميل كار قبلي خودش اقدام مي‌نمايد. در فصل بعد نتيجة آزمايشهاي انجام شده در مورد هر روش به تفصيل نشان داده شده است.

۷- آزمايشها و نتايج

در اين فصل، آزمايشهايي كه بر روي هر يك از روشها به منظور بررسي نقاط ضعف و قوت آنها انجام گرفته است به همراه نتايج آنها نشان داده مي‌شوند.

۷-۱- نحوة طراحي آزمايشها

در فصل پنجم كه به شرح سيستم آزمون مي‌پردازد، عاملها از لحاظ ويژگي‌هاي رفتاري مانند قابليت اعتماد، ضريب اهميت وظيفه، حجم بار كاري و ... به چند دسته مجزا تفكيك شدند و در اينجا طراحي الگوهاي خرابي براي هر عامل با انتخاب آنها از همين دسته‌ها انجام شده است.

آزمايشهاي انجام شده شامل ۳۰ الگوي خرابي است كه ۲۰ الگوي اول به صورت تصادفي توليد شده‌اند ولي ۱۰ الگوي بعدي، موارد خاص و نسبتا دشوارتري هستند كه در مقايسه با ۲۰ مورد قبلي با احتمال بسيار كمتري امكان دارد كه در محيط عملياتي رخ بدهند و به صورت دستي طراحي شده‌اند. الگوهاي خرابي تصادفي از لحاظ رخداد يا عدم رخداد خطا براي عاملها و نيز طول خطاي آنها، با توجه به ميزان قابليت اعتماد عاملها طراحي شده‌اند، براي مثال عامل با قابليت‌اعتماد نسبي ۳ با احتمال ۰,۳۳ و عامل با قابليت اعتماد ۲ با احتمال ۰,۵ دچار خرابي مي‌شوند. به همين ترتيب طول خرابي عامل با قابليت اعتماد بالا، كوتاهتر از خرابي عامل با قابليت اعتماد پايين است.

با توجه به ضريب قابليت‌اعتمادي كه براي عاملها در نظر گرفته‌ايم، احتمال رخداد هر يك از آزمايشهاي تصادفي به طور متوسط $\left(\frac{1}{R_i}\right)^3$ است كه بيانگر سه خرابي براي عاملهاي سيستم است، اما الگوهاي خاص احتمال رخدادي به مراتب كمتر، به‌طور متوسط در حد $\left(\frac{1}{R_i}\right)^6$ دارند.

البته به دليل تصادفي بودن، مجموعة آزمايشهاي تصادفي گاه شامل الگوهاي ساده تري هم هست كه جلوتر به اين مساله اشاره شده‌است. شرح اين الگوهاي خرابي كه در نمودارهاي اين فصل، با موارد ۱ تا ۱۱ به آنها اشاره شده‌است، در اينجا آورده شده‌است:

۱) مقدار متوسط خرابي‌هاي تصادفي (در ۲۰ آزمايش با الگوهاي تصادفي)
۲) خرابي همة عاملها به جز سه عامل، با قابليت اطمينان بالا، با قابليت اطمينان پايين و خيلي سريع با هم: **عاملهاي ۱ و ۵ و ۱۰ سالم هستند.**
۳) خرابي همة عاملهاي قابل اعتماد با هم: **عاملهاي ۶، ۷، ۸، ۹ و ۱۰ خراب مي شوند.**
٤) خرابي همة عاملهاي غيرقابل اعتمادتر با هم: **عاملهاي ۱، ۲، ۳، ۴و ۵ خراب مي شوند.**
٥) خرابي دو عامل پر اهميت و دو عامل با اهميت متوسط با هم : **عاملهاي ۶، ۷، ۹ و ۱۰ خراب مي شوند.**
٦) خرابي دو تا از مهمترين عامل ها با هم: **عاملهاي ۹ و ۱۰ خراب مي شوند.**
۷) خرابي همة عاملهاي كم اهميتتر با هم: **عاملهاي ۱، ۲، ۳، ۴ و ۵ خراب مي شوند.**
۸) خرابي همة عاملها به جز سه عامل سريع و متوسط و كند با هم: **فقط عاملهاي ۵، ۶ و ۷ سالم هستند.**
۹) خرابي تدريجي عاملهاي مهم: **عاملهاي ۶، ۷، ۸، ۹ و ۱۰ خراب مي شوند.**
۱۰) خرابي تدريجي عاملهاي سريع: **عاملهاي ۵ و ۸ خراب مي شوند.**
۱۱) خرابي تدريجي عاملهاي كم اهميت تر: **عاملهاي ۱، ۲، ۳، ۴ و ۵ خراب مي شوند.**

در مورد هر يك از الگوهاي خرابي كه در بالا شرح داده شد، ٤ وضعيت براي باس ورودي سيستم كه توسط Frame Generator تغذيه مي‌شود و دادة مورد نياز عاملها را براي انجام وظيفه‌شان در اختيارشان قرار مي‌دهد، در نظر گرفته ايم.

البته در مورد روشهاي ۱ و۲، تنها از باس قطعي استفاده شده است ولي ديگر روشها در هر ٤ وضعيت بررسي شده اند.

۱) حالت قطعي[1]:
۲) در اين حالت پريود انتساب وظيفه برابر ۲۰ است و عاملها نيز از اين مساله كاملا مطلع هستند.

1 Deterministic Situation

۳) حالت غير قطعي با درجة عدم قطعيت پايين[1]:
٤) در اين حالت ميانگين پريود انتساب وظيفه ۲۰ است ولي انحراف معيارآن برابر است با ۱,٥. عاملها نيز از هر دو مقدار ميانگين و انحراف معيار به طور كامل مطلع هستند.
٥) حالت غير قطعي با درجة عدم قطعيت متوسط[2]:
٦) در اين حالت ميانگين پريود انتساب وظيفه ۲۰ است ولي انحراف معيار آن برابر است با ۳. و باز هم عاملها نيز از هر دو مقدار ميانگين و انحراف معيار به طور كامل مطلع هستند.
۷) حالت غير قطعي با درجة عدم قطعيت بالا[3]:
۸) در اين حالت ميانگين پريود انتساب وظيفه ۲۰ است ولي واريانس آن برابر است با ٦ و مانند دو حالت قبل عاملها نيز از هر دو مقدار ميانگين و انحراف معيار به طور كامل مطلع هستند.

در جدول زير نشان داده شده است كه كداميك از حالات بالا براي هر روش پيشنهادي مورد بررسي قرار گرفته است:

جدول 2. روشها و نوع باس اعمال شده براي تست به هر كدام

	Deterministic	Low NonDet.	Medium NonDet.	Highly NonDet.
روش اول	+	-	-	-
روش دوم	+	-	-	-
روش سوم	+	+	+	+
روش چهارم	-	+	+	+
روش پنجم	-	+	+	+

۷-۲- معيارهاي ارزيابي

۷-۲-۱- كارايي

كارايي، رايجترين معيار ارزيابي عملكرد هر سيستمي در هر حالتي اعم از قطعي يا غيرقطعي ميباشد.

1 Low Non-Deterministic Situation
2 Medium Non-Deterministic Situation
3 Highly Non-Deterministic Situation

براي ارزيابي كارايي اين سيستم، تعداد وظيفه‌هايي را كه با موفقيت به انجام رسيده‌اند با احتساب درجه اهميت آنها براي سيستم در مقايسه با كل وظيفه‌هايي كه به عاملها محول شده‌اند، در نظر مي‌گيريم:

$$Performance = \frac{1}{\sum_{j=1}^{M}\sum_{i=1}^{N} C_{ij}} \sum_{j=1}^{M}\sum_{i=1}^{n} C_{ij} \qquad (۱)$$

كه در اين فرمول، n تعداد وظيفه‌هايي است كه با موفقيت تكميل شده‌اند و ضريب بحران آنها C_{ij} است، N تعداد كل وظيفه‌ها و M تعداد ضرايب بحران مختلفي است كه براي كار مجموعة عاملهاي مختلف موجود در سيستم تعريف شده است.

۷-۲-۲- متوسط زمان پاسخ وزندار[1]

براي ارزيابي كيفيت زماني اجرا و تكميل وظيفه‌ها، نيازمند معياري هستيم كه به نحوي بيانگر زمان پاسخ سيستم باشد، به اين معنا كه با احتساب اهميت وظيفة تكميل‌شده براي سيستم و نيز تاخير ميان زمان انتساب وظيفه تا تكميل آن، شناختي از نحوة عملكرد طراحي‌هاي مختلف در مقايسه با يكديگر به‌دست آوريم. به همين منظور، معياري با نام متوسط زمان پاسخ وزندار تعريف شده است. اين معيار به صورت زير تعريف مي‌شود:

$$AWRT = \frac{1}{AgentsNo} \sum_{j=1}^{AgentsNo} \frac{1}{n}\sum_{i=1}^{n} C_i T_i \qquad (۲)$$

كه در اين فرمول $AgentsNo$ تعداد عاملهاي موجود در سيستم، n تعداد وظيفه‌هاي انجام شده، C_i ضريب بحران وظيفة تكميل شده و T_i كه به صورت زير تعريف مي‌شود نشاندهندة ميزان تاخير(زمان سپري شده) از لحظة محول شدن وظيفه به يك عامل (اعم از اينكه عامل در حال انجام وظيفة حالت عادي خود باشد يا اينكه وظيفه‌اي را به منظور كمك‌رساني بر عهده گرفته باشد) تا لحظة تكميل آن است.

$$T_i = CompletionTime_i - AssignmentTime_i$$

1 Average Weighted Response Time

۷-۲-۳- متوسط فاصله زماني وزندار تكميل تا مهلت مقرر[1]

علاوه بر معياري كه در بالا با نام متوسط زمان پاسخ وزن‌دار معرفي شد، معيار زماني ديگري نيز در اينجا معرفي شده است كه به مقدار زمان باقيمانده تا تكميل وظيفه، پيش از موعد مقرر، توجه مي‌كند و با احتساب درجه اهميت وظيفه‌هاي تكميل شده پيش از موعد مقرر، يك معيار زماني وزن‌دار ارائه مي‌كند. اين معيار به اين ترتيب تعريف مي‌شود:

$$AWAOD = \frac{1}{AgentsNo}\sum_{j=1}^{AgentsNo}\frac{1}{n}\sum_{i=1}^{n} C_i.AheadTime_i \qquad (۳)$$

كه در اين فرمول $AheadTime_i = Deadline_i - CompletionTime_i$ است. اين معيار به همراه معيار پاسخ زماني كه در بالا بيان شد، بيانگر سرعت سيستم و كيفيت زماني روشها در اجراي وظيفه‌هاست.

۷-۲-٤- متوسط عدم هماهنگي كمك و كمك‌رسان[2]

به منظور ارزيابي روشها از نظر كيفيت تقسيم وظيفة انجام شده، معياري معرفي شده است تا از اين طريق تعيين شود كه كدام روش مناسبترين وظيفه‌ها را با توجه به قابليت‌هاي عاملها به آنان منسوب نموده است. بنابراين، معيار حاضر نشان‌دهندة كيفيت تقسيم وظيفه‌هاست: هرچه كارهاي مهمتر توسط عاملهاي قابل اعتمادتر انجام شده باشد، نشان از كيفيت بهتر كار دارد.

اين معيار به صورت زير تعريف مي‌شود:

$$AHInc = \left|1 - \frac{1}{AgentsNo}\sum_{j=1}^{AgentsNo}\frac{1}{R_j n_j}\sum_{i=1}^{n_j} C_i\right| \qquad (۴)$$

كه در آن n_j تعداد وظيفه‌هاي تكميل شده با ضريب بحران C_i توسط عاملهايي با قابليت اطمينان Rj را نشان مي‌دهد. مطلوب آن است كه اين مقدار نزديك به صفر باشد، چنانكه اگر عاملي همواره تنها وظيفة خودش را كه هماهنگترين وظيفه با قابليتهاي اوست انجام بدهد، اين مقدار به مي‌نيمم مي‌رسد.

1 Average Weighted Ahead of Deadline Completion Time(AWAOD)
2 Average Help-Helper Incompatibility (AHInc)

۷-۲-۵- قابليت كنارآمدن سريع با خرابي‌هاي آينده[1]

هنگامي كه عاملها به مرحلة بررسي ليست كمكهاي موردنياز مي‌رسند، اگر بر اساس منطقي كه براي هر روش اتخاذ شده است، به يك عامل هيچ وظيفه‌اي تخصيص نيابد يا به عبارتي انجام هيچ وظيفه‌اي را به عنوان كمك تقبل ننمايد، و از طرفي زمان آزادي كه اين عامل تا رسيدن وظيفة بعدي خودش در اختيار دارد به اندازة انجام اقلا يك وظيفه از وظايف موجود سيستم (البته با احتساب توان پردازشي هر عامل) باشد، اين عامل به عنوان يك نقطة قوت و اتكا براي سيستم باقي مي‌ماند تا سيستم بتواند در صورت بروز خرابي احتمالي در آينده براي هر يك از عاملهاي ديگر كه در حال حاضر سالم هستند، سريعا از عهدة انجام وظيفه برآيد. بنابراين در نظرگرفتن تعداد عاملهايي كه كاري به آنها تخصيص نيافته است و در ضمن از مدت زمان كافي (زمان آزادي آنها اقلا به اندازة يك كمك با توانايي خودشان به يكي از هم‌تيمي‌هايشان است) برخوردار هستند به همراه ضريب قابليت اطمينان آنها، مي‌تواند معياري براي ارزيابي كيفيت تقسيم وظيفة سيستم از لحاظ كمك‌رساني در آينده باشد. با توجه به اينكه همة عاملهاي ما در سيستم، از توانايي‌هاي يكساني برخوردار نيستند و براي تعدادي از آنها ويژگي برخورداري از قابليت ويژه در نظر گرفته شده است، عاملهاي كمك‌رسان معمولي[2] و عاملهاي خاص[3] را در ارزيابي اين مساله از هم جدا كرده‌ايم. تعريف اين معيارها به صورت زير است:

$$\text{IFFC-Ordinary} = \frac{1}{AgentsNo} \sum_{i=1}^{AgentsNo} Fo_i.R_i \qquad (۵)$$

$$\text{IFFC-Special} = \frac{1}{AgentsNo} \sum_{i=1}^{AgentsNo} Fs_i.R_i \qquad (۶)$$

كه در اين دو فرمول Fo_i و Fs_i به ترتيب تعداد دفعاتي است كه عاملهاي معمولي يا خاص با ضريب قابليت اطمينان R_i با داشتن وقت كافي آزاد مانده‌اند.

1 Immediate Future Fault Compensation(IFFC)
2 Ordinary Heper Agents
3 Special Helper Agents

اكنون در جدول زير نشان داده شده است كه كداميك از معيارهاي ارزيابي براي كداميك از روشهاي سيستم مورد محاسبه قرار گرفته است:

جدول 3. روشها به همراه معيارهاي ارزيابي محاسبه شده براي هر كدام

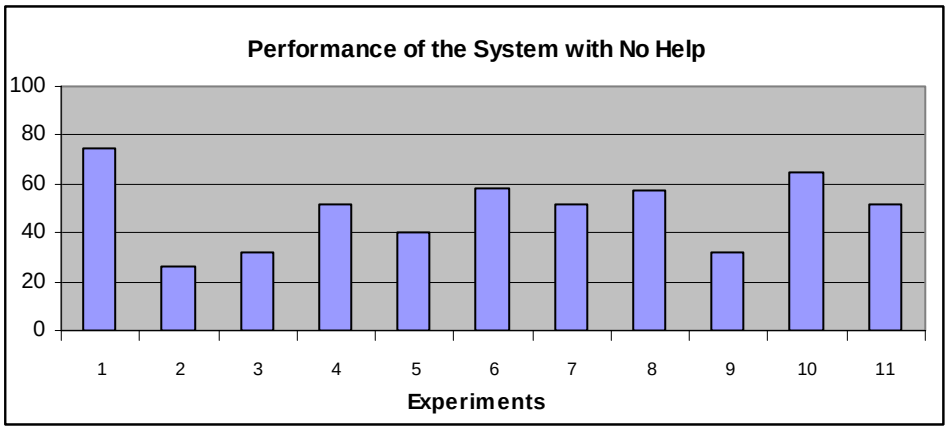
	Performance	AWRT	AWAOD	AHInc	IFFC
روش اول	+	+	+	+	+
روش دوم	+	+	+	+	+
روش سوم	+	+	+	+	+
روش چهارم	+	+	+	+	-
روش پنجم	+	+	+	+	-

همانطور كه در جدول بالا مشاهده مي‌كنيد، روش چهارم و پنجم به دليل غيرقطعي بودن اطلاعاتي از زمان آزادي خود تا رسيدن وظيفة بعدي‌شان ندارد و در نتيجه محاسبة IFFC در مورد آنها معنايي ندارد، مگر اينكه اين محاسبه با نايقيني و با احتساب ميانگين پريود وظيفه‌ها به جاي پريود دقيق انجام گيرد.

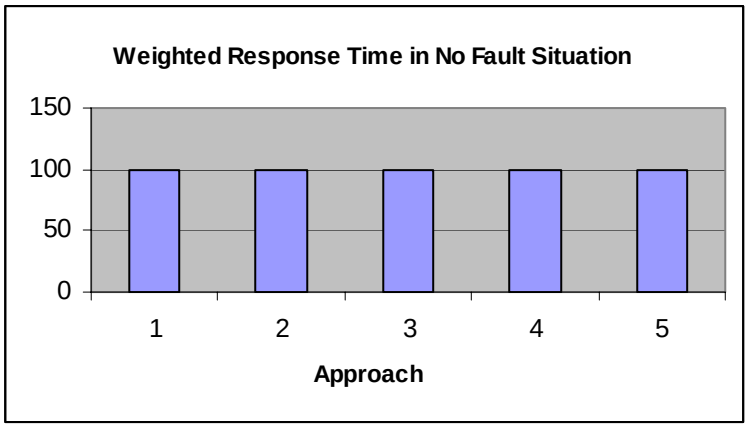
در اينجا پيش از اينكه به بيان آزمايشها بپردازيم ذكر يك نكته ضروري است. براي مقايسة صحيح و منطقي روشهاي پيشنهادي با توجه به معيارهاي بالا بايد ابتدا در اهداف سيستم تامل كرده، آنها را اولويت بندي نمود. مثلا همواره كارايي را اولويت اول دانسته ، سپس زمان پاسخ را مهم بدانيم. يا اينكه ابتدا معيارهاي زماني و سپس هدفي چون تحمل پذيري خطا را مد نظر داشته باشيم. اين مساله بستگي مستقيم به كاربرد سيستم دارد. ما در اين پروژه به دليل اهميتي كه براي ايجاد تحمل پذيري خطا به عنوان تاكيد اصلي وجود داشته است، پس از كارايي (ماكزيمم نمودن تعداد وظيفه هايي كه پيش از مهلت مقرر خود به انجام مي رسند)، به مسالة تحمل پذيري خطا توجه داريم. بديهي است كه براي انتقال ايده هاي اين تحقيق به سيستمهايي با ذات سريال يا به طور كلي نيازمند پاسخ زماني كوتاه، قطعا يا ترتيب ديگري از روشها حاصل مي گردد يا در روشهاي فعلي بايد تغييراتي صورت بگيرد .

۷-۳- شرح آزمايشها

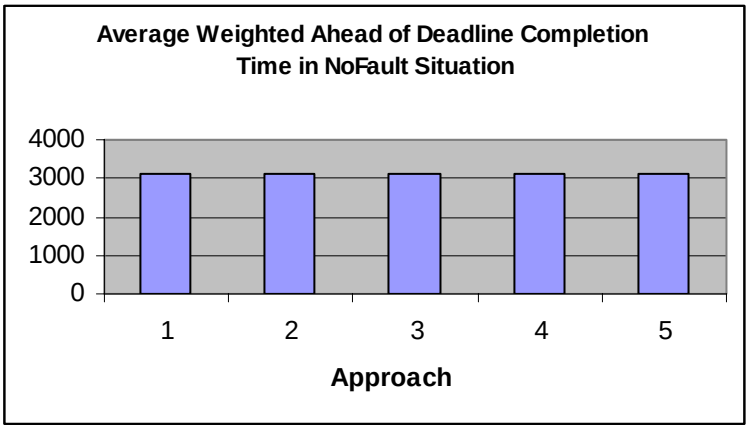
در اين قسمت به نحوة ارزيابي هر روش مي‌پردازيم و در آخر مقايسه‌اي اجمالي از روشهاي پيشنهادي در اين تحقيق ارائه مي‌كنيم. در مورد هر يك از روشها، الگوهاي خرابي در حالت مناسبي از باس ورودي به سيستم اعمال شده اند و نمودارهاي ارزيابي عملكرد سيستم نشان داده شده است. در اغلب نمودارها محور افقي، يا آزمايشهاي انجام شده و در چند مورد ديگر روشهاي پيشنهادي را نشان مي دهد. شكل14 نمودار كارايي سيستم در حالت بدون كمك را نشان مي‌دهد. به علاوه، نمودارهاي پاسخ زماني و فاصله زماني تكميل وظيفه تا مهلت زماني مقرر در شكل15 و شكل16 آمده اند. نمودار عدم سازگاري كمك و كمك‌رسان به دليل همواره صفر بودن در شرايط بدون خطا رسم نشده است. در ضمن نمودار مربوط به كنارآمدن با خرابي‌هاي آينده، تنها در صورت بروز خرابي و پس از تقسيم وظيفه براي كمك‌رساني محاسبه مي‌شود و بنابراين در اينجا رسم نشده است.



شكل 14- كارايي سيستم در حالت بدون كمك



شكل 15 - پاسخ زماني سيستم با روشهاي پيشنهادي در حالت بدون خطا



شكل 16-فاصلة زماني تكميل وظيفه تا مهلت زماني مقرر سيستم در روشهاي پيشنهادي در حالت بدون خطا

۷-۳-۱- روشهاي قطعي

روشهاي قطعي شامل روشهاي ۱،۲و۳ هستند كه در اين قسمت به آزمايشهاي انجام شده در مورد آنها اشاره مي‌شود.

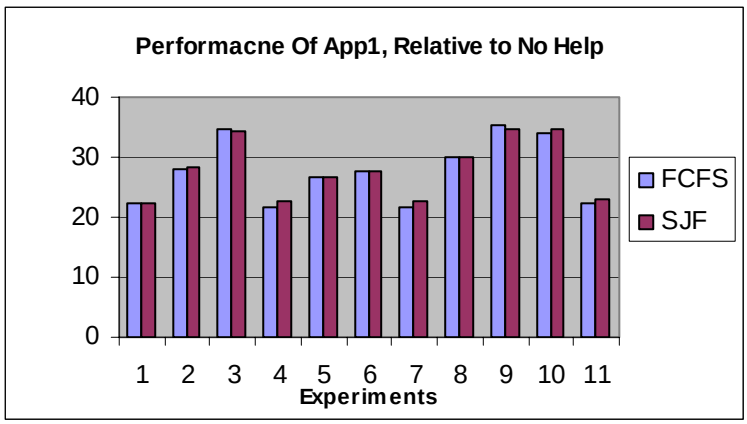
۷-۳-۱-۱- روش اول

در مورد روش اول، دو سياست SJF-over-FPF و FCFS-over-FPF مورد ارزيابي قرار گرفته‌اند. همانطور كه در نمودارهاي شكل17 تا شكل22 مشاهده مي‌شود، تنها تفاوت مهم اين دو سياست، در فاصلة زماني تكميل وظيفه تا مهلت زماني در مواردي از آزمايشها با الگوي خاص خرابي است (و نه آزمايشهاي تصادفي كه محتمل تر هستند) كه در آنها به دليل حساسيت زياد نحوة توزيع وقت كمك‌رسان، با SJF عملكرد مناسبتري حاصل شده است. به عبارت خلاصه‌تر، چنانكه شكل20 نشان مي‌دهد، تفاوت اين دو سياست تنها

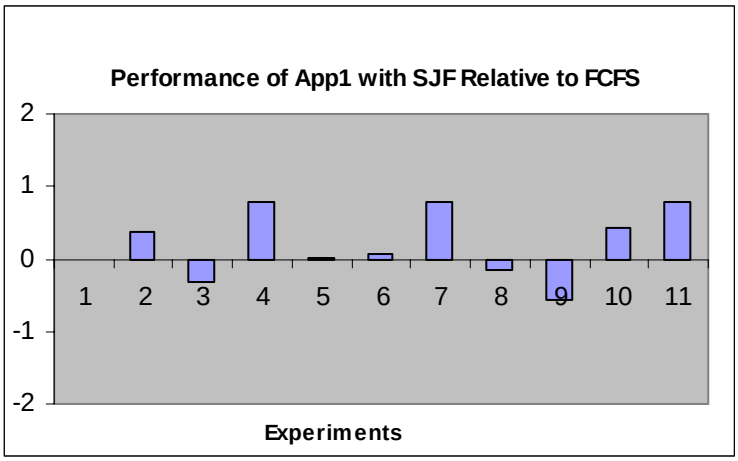
در کیفیت زمانی عملکرد سیستم آن هم در موارد خاصی است که در آنها تعداد کمک‌رسانها از تعداد کمک‌های موردتقاضا کمتر است و سرعت تکمیل وظایف با سیاست اولویت به کوتاهترین وظیفه، افزایش می‌یابد. از لحاظ کارایی، کنارآمدن با خرابی‌های آینده و سازگاری کمک با کمک‌رسان هر دو سیاست پیشنهادی در این روش دارای نقایصی هستند که در ارائهٔ روشهای بعدی سعی در رفع آنها شده است.

همانطور که از نمودار کارایی در شکل17 برمی‌آید، کارایی در اغلب موارد نسبت به حالت بدون کمک، افزایش قابل قبولی داشته است (از ۲۰درصد در آزمایشهای ساده‌تر تا ۳۵ درصد در آزمایشهای مشکلتر). چنانکه شکل19 نشان می‌دهد، پاسخ زمانی سیستم با بکارگیری سیاست SJF نسبت به FCFS بهبود می‌یابد که این به ویژگی SJF در بهینه‌سازی زمان‌پاسخ مربوط می‌شود. در مورد عدم سازگاری کمک و کمک‌رسان که در شکل21 دیده می‌شود، ذکر این نکته مهم است که در روش اول هیچگونه تمهیدی اندیشیده نشده است و عاملها بدون توجه به اهمیت وظیفهٔ مورد تقاضا برای سیستم و قابلیت اعتماد خودشان، اقدام به کمک‌رسانی می‌کنند که این موجب بروز ناسازگاری زیاد بر طبق تعریف فرمول معیار AHInc می‌گردد.

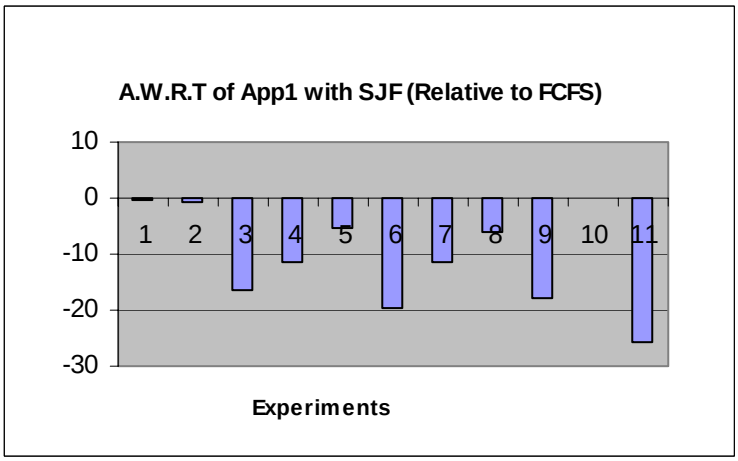
با توجه به شکل22 مشاهده می‌شود که به دلیل نپرداختن به مسالهٔ تقسیم صریح وظیفه در روش اول و عدم توجه عاملهای کمک‌رسان به نیازمندی‌های کمک مورد تقاضا و قابلیتهای خودشان، این روش در مقایسه با روشهای بعدی از ضعف قابل‌ملاحظه‌ای برخوردار است.



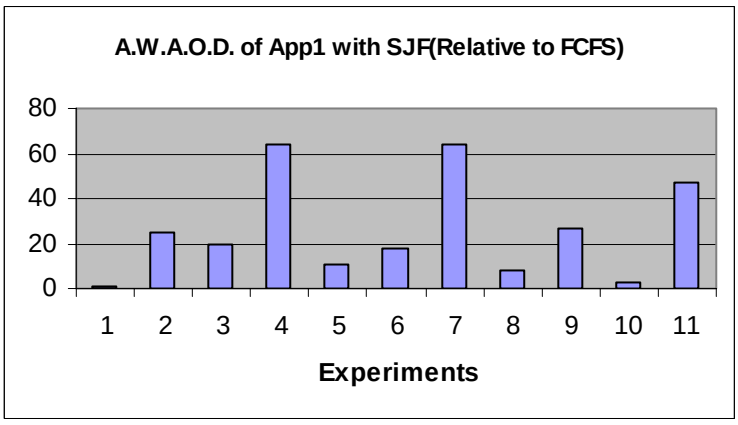
شکل17 - کارایی روش اول در مقایسه با حالت بدون کمک



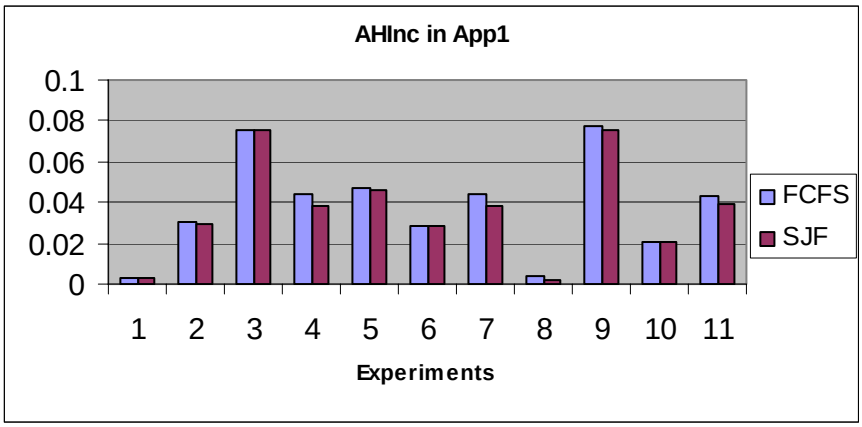
شکل18 - مقایسة کارایي دو سیاست FCFS و SJF در روش اول از لحاظ کارایي



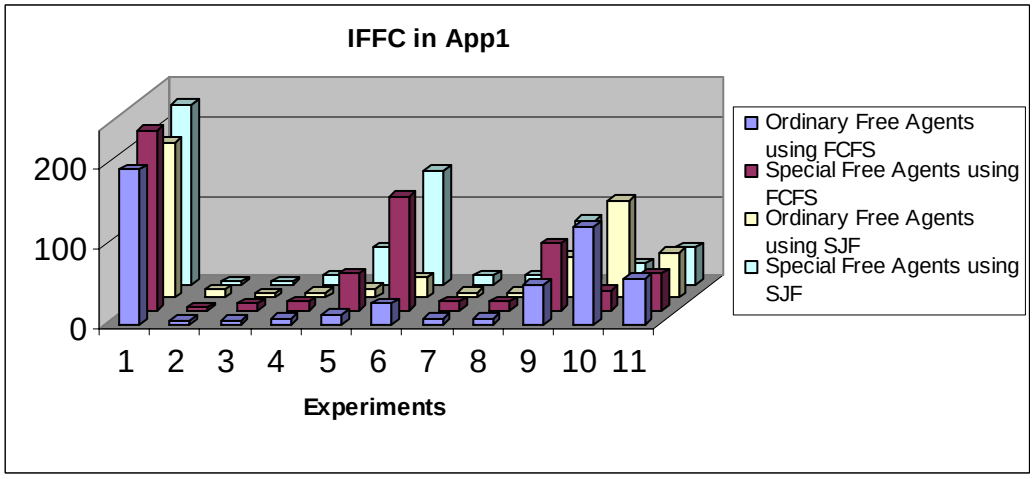
شکل19. پاسخ زماني روش اول با سیاست SJF در مقایسه با FCFS



شکل20. فاصله زماني تکمیل تا مهلت مقرر در روش اول با سیاست SJF در مقایسه با FCFS



شکل21. عدم سازگاري کمك و کمك‌رسان در روش اول



شکل 22. کنارآمدن با خرابي‌هاي آينده در روش اول

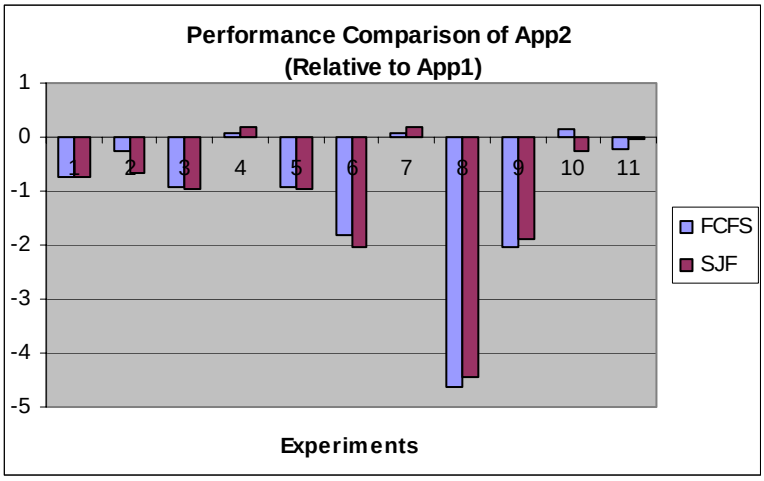
۷-۳-۱-۲-روش دوم و مقايسة آن با روش اول

در مورد روش دوم نيز، دو سياست SJF-over-FPF و FCFS-over-FPF مورد ارزيابي قرار گرفته‌اند. همانطور كه ارزيابي نمودارها نشان مي‌دهد، اين دو سياست نيز تفاوت عمده‌اي با هم ندارند، مگر در موارد خاصي از الگوهاي خرابي مشابه روش اول. اما آنچه روش دوم را از روش اول مستثني مي‌كند، بهبود نسبي دو معيار ارزيابي آخر يعني عدم سازگاري كمك و كمك‌رسان و كنارآمدن با خرابي‌هاي آينده است كه اين مقايسه به همراه مقايسة ديگر معيارها در نمودارهاي شكل23 تا شكل27 آمده است.

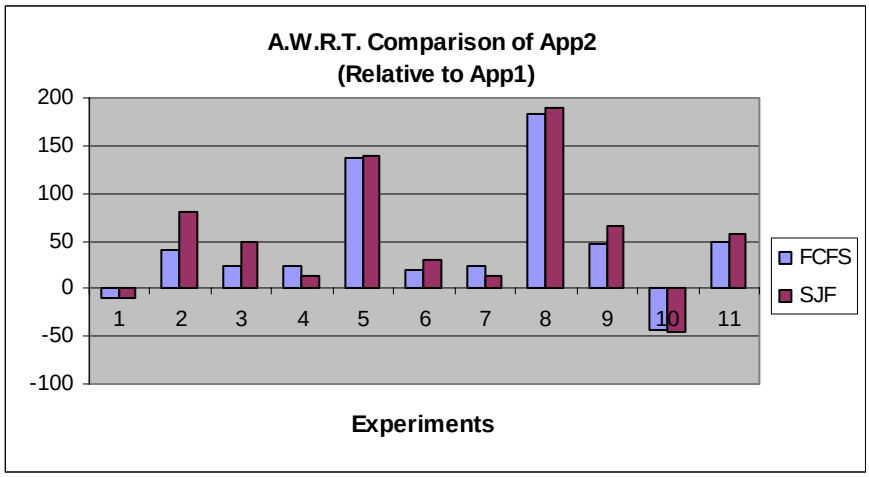
همانطور كه مشاهده مي‌شود، بهترين سياست از لحاظ كارايي و به ويژه پاسخ زماني و فاصلة زماني تكميل تا مهلت مقركه در شكل24 و شكل25 نشان داده شده‌اند، در اغلب آزمايشها، به‌كارگيري سياست SJF است. با نگاه كلي به شكل23 مشاهده مي‌شود كه به خاطر نحوة طراحي

روش پيشنهادي دوم، در برخي موارد عاملها از بيم ناهماهنگي با كمك مورد نياز، از كمك‌رساني امتناع كرده‌اند كه اين موجب كاهش كارايي دراغلب موارد شده است. ولي در عوض همانطور كه شكل26 نشان مي‌دهد، ناهماهنگي ميان كمك و كمك‌رسان در روش دوم در مقايسه با روش اول به مي‌نيمم رسيده است و اين به خاطر طراحي خاص روش دوم است كه در آن به اين مساله بهاي زيادي داده شده‌است كه همواره عاملها به تقاضاهايي پاسخ مثبت بدهند كه بيشترين هماهنگي را با توانايي‌هاي آنها دارد و تقاضاهاي ديگر را براي ديگر هم‌تيمي‌هاي خود باقي بگذارند. با توجه به همين مساله، شكل27 نشان مي‌دهد كه عاملهاي كمك‌رسان بيشتري چه معمولي و چه با قابليت ويژه، در اين روش آزاد مانده‌اند و اين خود به عنوان حسن مهمي براي اين روش به حساب مي‌آيد.

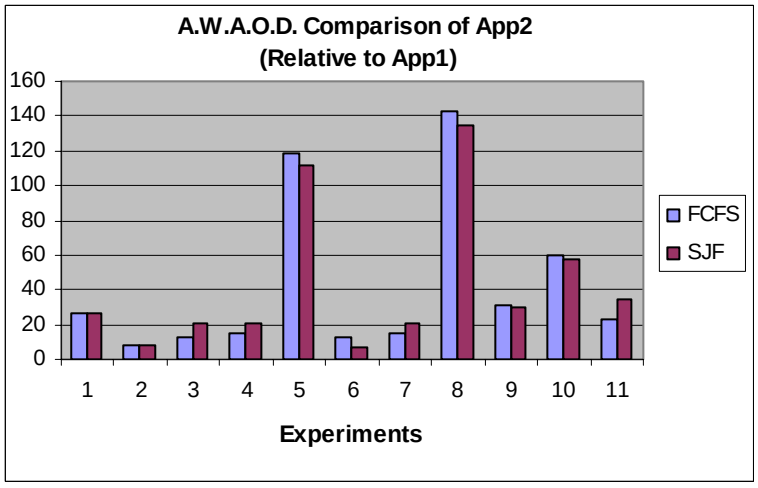
در ادامه، با ارائة روش سوم نقص موجود در كارايي و زمان پاسخ برطرف گرديده و كنارآمدن با خرابي‌هاي آينده به ميزان قابل توجهي بهبود مي‌يابد. لازم به توضيح است كه، در مقايسه‌هايي كه در بخش‌هاي بعدي انجام مي‌شود، از ميان دو سياست موجود در روشهاي ۱ و ۲، از نتايج سياست SJF كه كارايي بهتري را نشان داده‌است، استفاده خواهد شد.



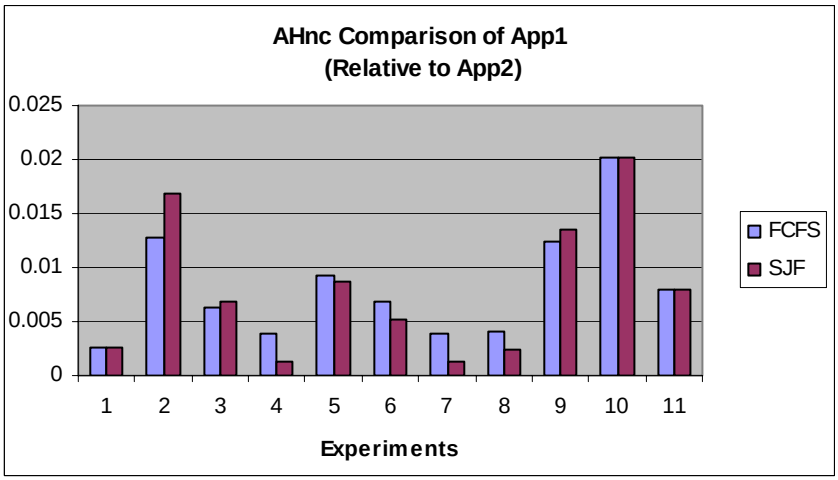
شكل 23. كارايي در روش دوم در مقايسه با سياستهاي متناظر در روش اول



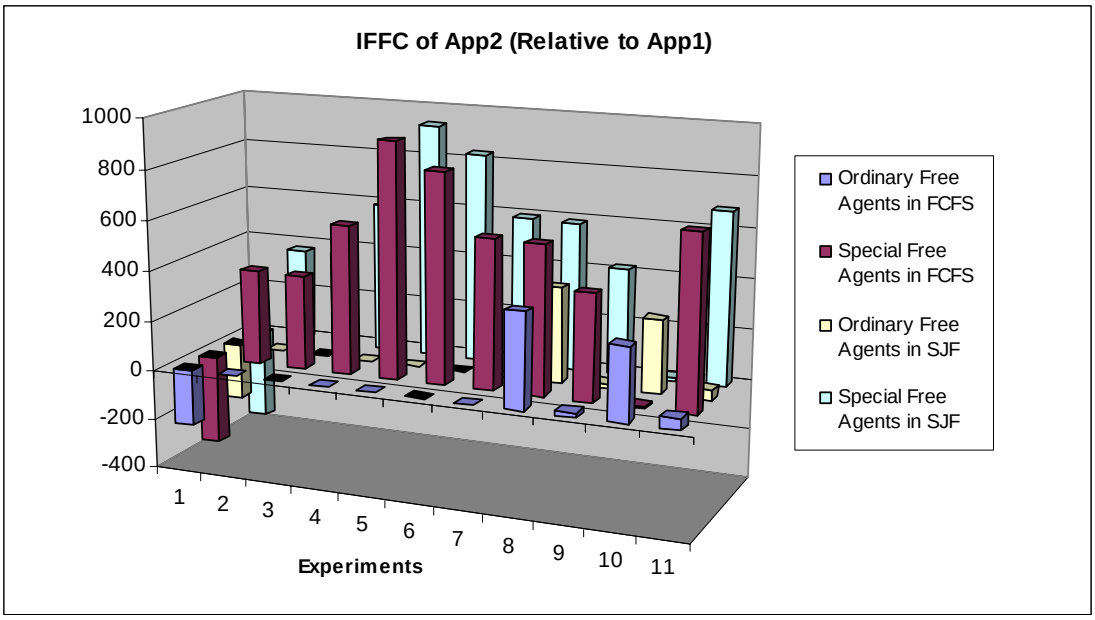
شکل 24. پاسخ زماني در روش دوم در مقايسه با سياستهاي متناظر در روش اول



شکل 25. فاصله زماني تكميل تا مهلت مقرر در روش دوم در مقايسه با سياستهاي متناظر در روش اول



شکل 26. عدم سازگاري كمك و كمك‌رسان در روش اول در مقايسه با سياستهاي متناظر در روش دوم



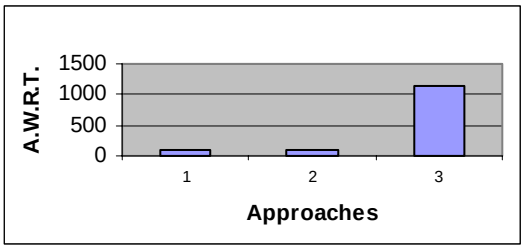
شكل 27. كنارآمدن با خرابي‌هاي آينده در روش دوم در مقايسه با سياست‌هاي متناظر در روش اول

۷-۳-۱-۳- روش سوم و مقايسة آن با دو روش اول

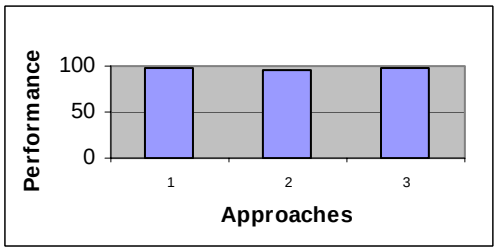
اين طرح كه به عنوان طرح تكميلي در روشهاي قطعي مطرح مي‌شود، به دليل بهره‌گيري از الگوريتم بهينة تقسيم وظيفه از قابليت‌هاي بسيار خوبي در مقايسه با دو روش سادة قبلي برخوردار است. نمودارهاي ارزيابي اين روش را در مقايسه با دو روش قبلي در شكل36 تا شكل41 مشاهده مي‌كنيد. همانطوركه نمودار شكل36 نشان مي‌دهد، كارايي روش سوم به ميزان قابل‌توجهي بالاتر از كارايي روشهاي اول و دوم است. البته نكتة جالب و قابل توجه در اين مقايسه آن است كه در آزمايشهايي كه روش سوم نتوانسته است كارايي را چندان افزايش بدهد،يا به عبارت صحيح‌تر نيازي به استفاده از الگوريتم بهينة تقسيم وظيفه نبوده است و روشهاي ساده‌تر هم از پس شرايط مساله به خوبي برآمده‌اند (مانند اغلب آزمايشهاي تصادفي و نيز آزمايش دهم)، پاسخ زماني توسط روش سوم افزايش يافته است و فاصله تكميل از مهلت زماني نيز طبيعتا كاهش يافته است. اين مساله در مورد الگوهاي تصادفي كه احتمال وقوعشان بيشتر است و به ويژه در مورد آزمايش دهم به خوبي ديده مي‌شود.شرح اين آزمايش براي يادآوري در اينجا آورده شده است:

خرابي عاملهاي ۵ و ۸ كه عاملهاي سريع سيستم هستند و بار كاري آنها از بقيه به ميزان قابل توجهي بيشتر است. اين آزمايش الگوي خرابي ساده اي است كه نياز خاصي به استفاده از الگوريتم تقسيم وظيفة بهينه ندارد و پيچيدگي نسبتا زياد اجراي اين الگوريتم بدون اينكه به بهبود كارايي بيانجامد، موجب افزايش زمان پاسخ شده است.

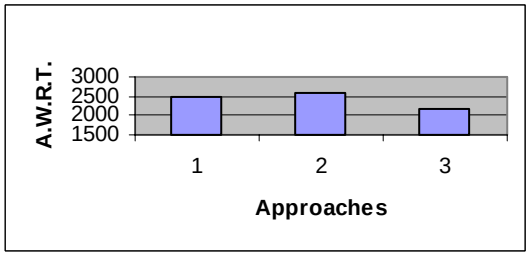
در حالي كه در مواردي مثل آزمايشهاي ۲، ۳و ٤ كه كارايي توسط روش سوم بهبود قابل ملاحظه اي داشته، پاسخ زماني هم بهبود مناسبي يافته است چرا كه طبق فرمولي كه در (۲)براي زمان پاسخ متوسط نشان داده‌شده است، پاسخ زماني براي وظيفه‌هاي تكميل شده محاسبه مي‌گردد، نه براي تمامي وظيفه‌ها و اين موجب بهبود زمان‌پاسخ روش سوم مي‌شود. اين مساله در نمودارهاي شكل28 تا شكل35 با دقت بيشتري نشان داده شده است.



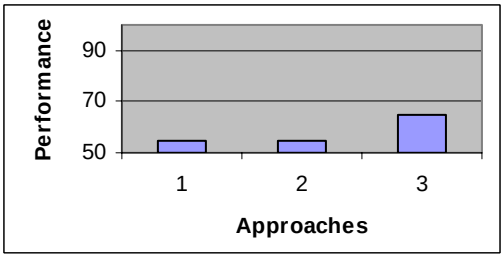
شکل 28. مقایسة زمان پاسخ متوسط وزن‌دار سه روش اول در آزمایش دهم که روش سوم موجب بروز زمان پاسخ طولاني شده است



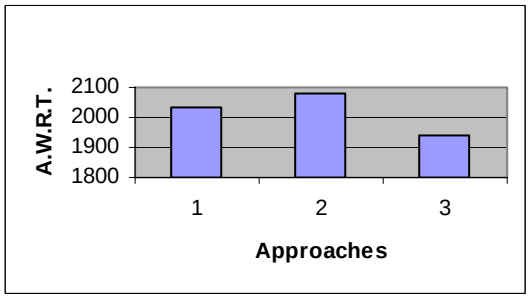
شکل 29. مقایسة کارایي سه روش اول در آزمایش دهم که روش سوم نتوانسته است تاثیرزیادي بر بهبود کارایي داشته باشد



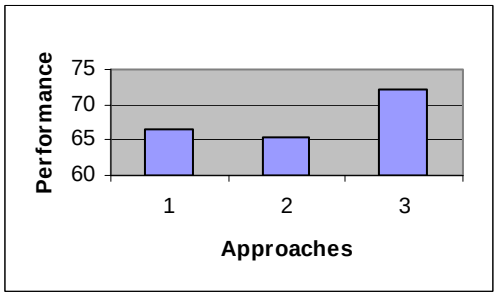
شکل30. مقایسة زمان پاسخ متوسط وزن‌دار سه روش اول در آزمایش ۲



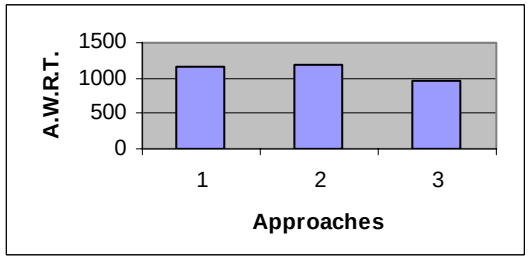
شکل 31. مقایسة کارایي سه روش اول در آزمایش ۲



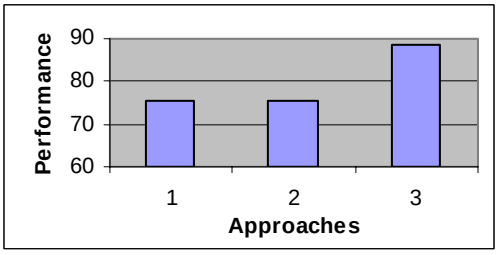
شکل 32. . مقایسة زمان پاسخ متوسط وزن‌دار سه روش اول در آزمایش۳



شکل 33. مقایسة کارایي سه روش اول در آزمایش ۳



شکل 34. . مقایسة زمان پاسخ متوسط وزن‌دار سه روش اول در آزمایش ٤



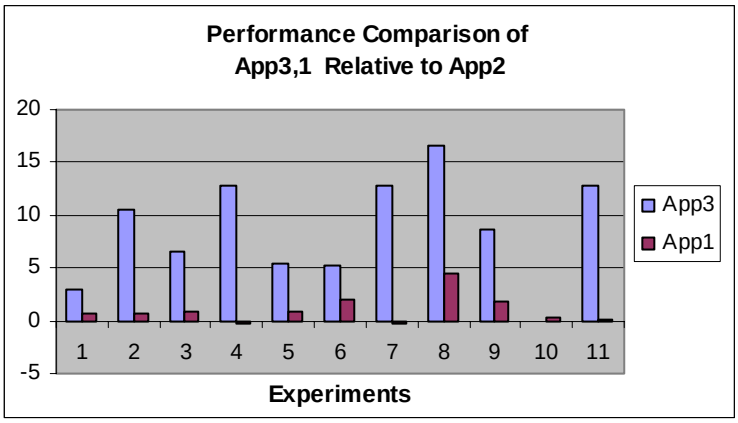
شکل35. مقایسة کارایي سه روش اول در آزمایش ٤

این مساله حاکي از پیچیدگي قابل توجه الگوریتم تقسیم بهینة وظیفه است که در صورتي که قادر نباشد از

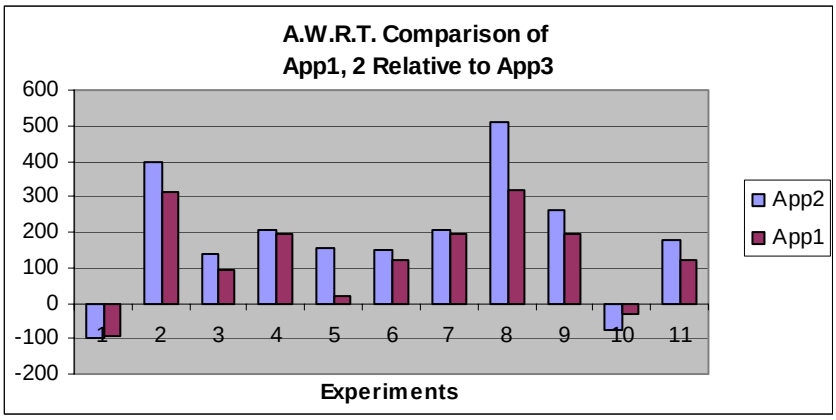
لحاظ بهبود كارايي مفيد واقع شود، ممكن است از لحاظ زمان پاسخ مشكلساز گردد. بنابراين چنانكه در مقايسة انتهاي اين فصل با احتساب احتمال رخداد هر يك از اين الگوهاي خرابي بر اساس قابليت اعتماد عاملها (براي محاسبة ميانگين وزن دار) نشان داده مي شود، روش سوم در مجموع با مشكل زمان پاسخ روبروست و اين موجب مي گردد كه اين روش در سيستم هاي گسترده اي كه قابليت اطمينان عملكرد عناصر آن قابل قبول است و خرابي هاي عمده اي در آن رخ نمي دهد، توصيه نشود.

در مورد معيار عدم هماهنگي ميان كمك و كمكرسان يادآوري اين مساله مهم است كه اين به بهاي آفزايش كارايي كه در روش سوم حاصل شده است بديهي است كه اين معيار اندكي افزايش داشته باشد.

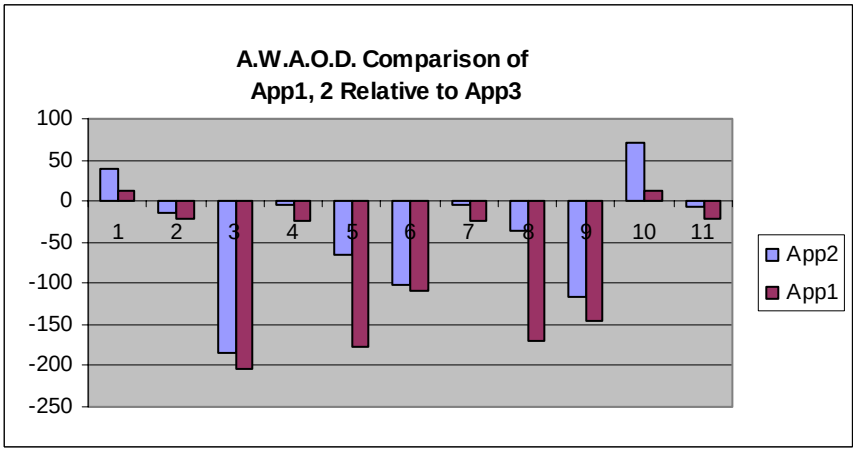
علاوه بر برتري هايي كه گفته شد، افزايش عاملهاي آزاد پس از تقسيم وظيفه، يكي از محاسن قابلتوجه در اين طرح است، چرا كه به علت تقسيم مناسب وظيفه ها بر اساس زمان آزادي هر عامل، سعي مي‌شود كه حتي‌الامكان مي‌نيمم تعداد عاملها درگير كمكرساني بشوند و اين مساله در شكل40 و شكل41 به وضوح ديده مي‌شود، هرچند كه بازهم به بهاي افزايش كارايي، در مواردي ديده مي‌شود كه عاملهاي آزاد براي آينده اندكي كاهش يافته اند.



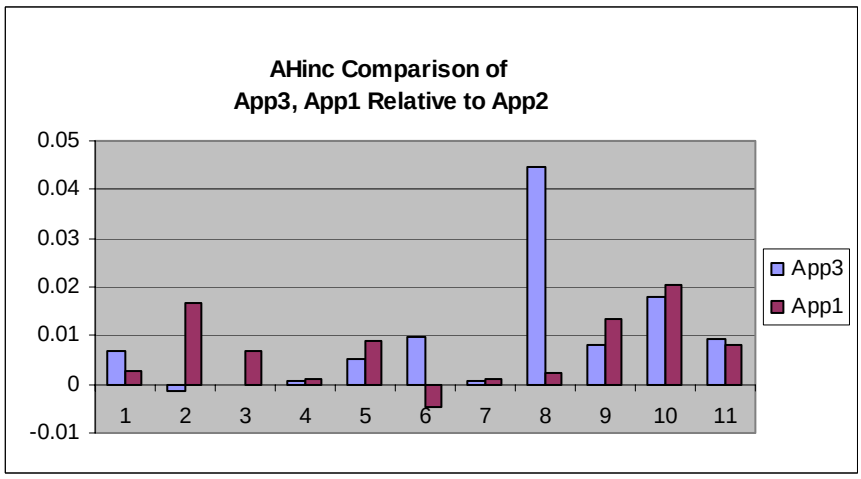
شكل 36. مقايسة كارايي سه روش اول



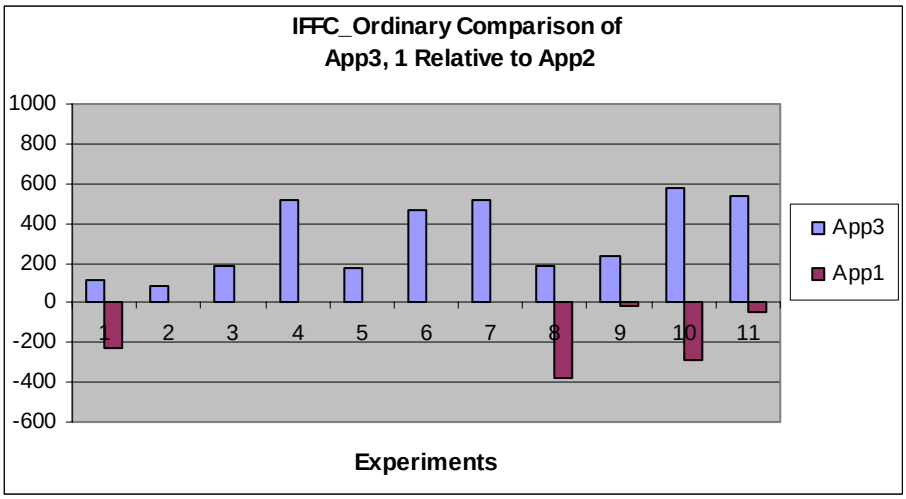
شکل 37. مقایسة پاسخ زماني سه روش اول



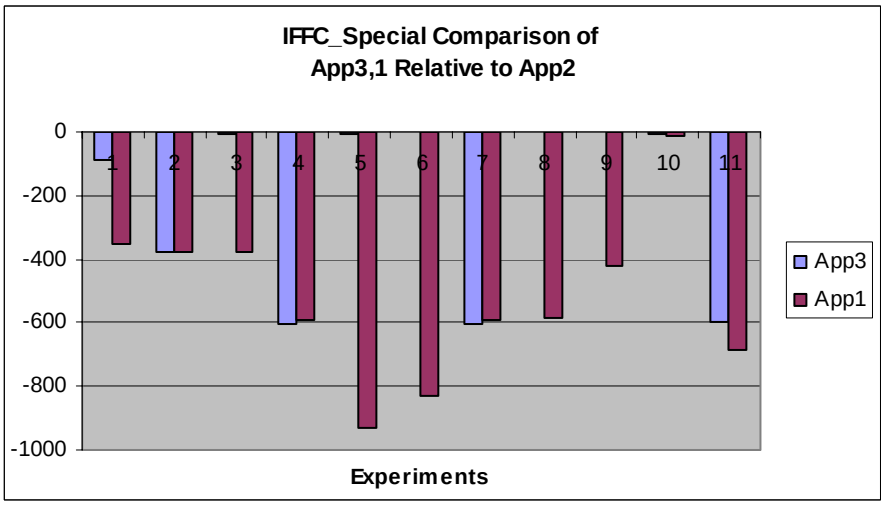
شکل38. مقایسة فاصلة زماني تکمیل تا مهلت مقرر در سه روش اول



شکل 39. مقایسة عدم سازگاري میان کمك و کمك‌رسان در سه روش اول



شكل 40. مقايسة كنارآمدن با خرابي‌هاي آينده به كمك عاملهاي معمولي در سه روش اول



شكل 41. مقايسة كنارآمدن با خرابي‌هاي آينده به كمك عاملهاي ويژه در سه روش اول

۷-۳-۲- روشهاي غير قطعي

روشهاي غيرقطعي شامل روشهاي ٤و ٥هستند كه در اين قسمت به آزمايشهاي انجام شده در مورد آنها اشاره مي‌شود.

۷-۳-۲-۱- روش چهارم

اين روش همانطور كه در بخش قبل اشاره شد، ويرايش غيرقطعي روش سوم است. يعني از تقسيم بهينة وظيفه‌ها براي تعيين كمك‌رسانهاي مناسب استفاده مي‌كند، ولي پيش از اجراي هر وظيفه امكان‌سنجي غيرقطعي در مورد آن وظيفه انجام مي‌دهد و با توجه به احتمال ريسك تعيين شده براي

او در زمان طراحي، تصميم مي‌گيرد كه كمك موردنظر را قبول نمايد يا خير.

در اين روش چهار حالت مختلف براي احتمال ريسك كردن عامل در نظر گرفته شده است كه اين ٤ حالت عبارتند از:

احتمال ريسك كم[1]: در حد ٠,١

احتمال ريسك زياد[2] :در حد ٠,٩

احتمال ريسك متوسط[3] : در حد ٠,٥

احتمال ريسك تطبيقي[4]: در صورتي كه كمك به عامل مهمي مدنظر باشد، مقدار ٠,٩، براي كمك به عامل كم اهميت مقدار ٠,١ و در مورد كمك به عامل با اهميت متوسط، از مقدار ٠,٥ براي احتمال ريسك استفاده شده است.

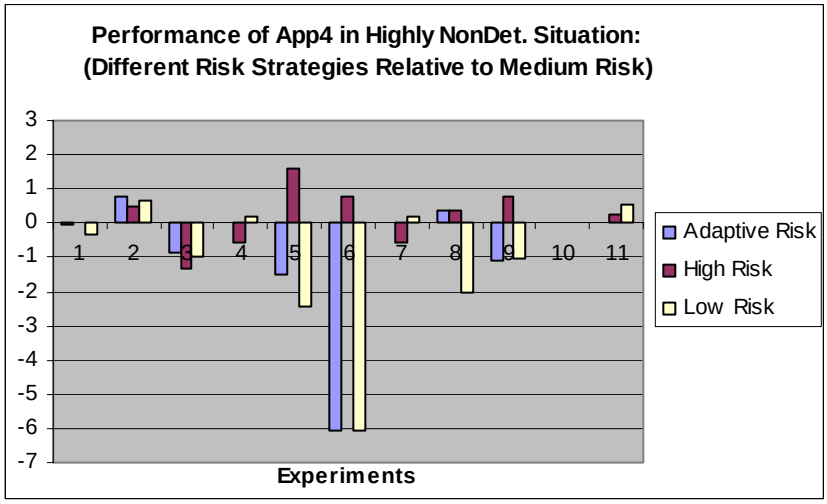
براي ارزيابي روش چهارم به عنوان يك سيستم غيرقطعي از هر سه حالت غيرقطعي كه براي باس ورودي سيستم متصور است نيز استفاده شده است تا عملكرد سيستم با افزايش عدم قطعيت هم مورد بررسي قرار بگيرد. لازم به تذكر است كه نمودارهاي اين بخش براي وضوح بيشتر، به طور نسبي ترسيم شده‌اند، يعني احتمال ريسك متوسط، به عنوان معيار مقايسه درنظر گرفته‌شده و كارايي سياست‌هاي ريسكي ديگر نسبت به اين سياست رسم شده‌اند.

٧-٣-٢-١-١- بررسي عملكرد روش چهارم با عدم قطعيت زياد در سيستم

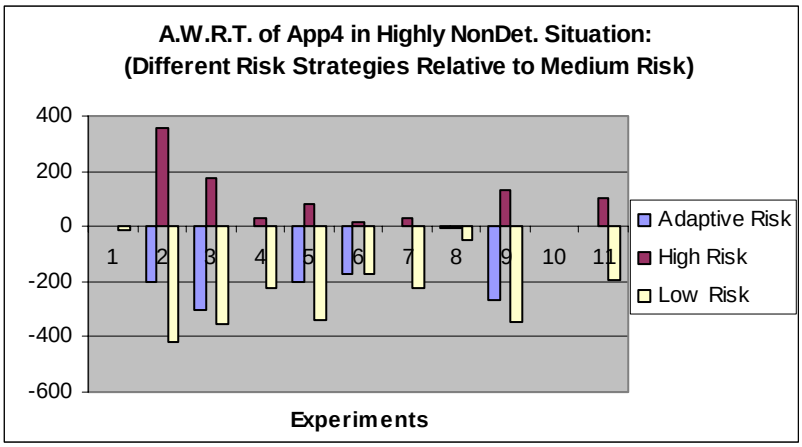
در اينجا نمودارهاي ارزيابي كارايي سيستم مبتني بر روش چهارم را با وجود عدم قطعيت زياد و با ٤ سطح مختلف از احتمال ريسك براي عامل‌هاي كمك‌رسان مشاهد مي‌كنيد. توجه به نمودارهاي ارزيابي كه درشكل42 تا شكل45 ديده مي‌شوند، تاييد مي‌كند كه از ميان اين چهار مقدار متفاوت براي احتمال ريسك، مناسبترين آنها احتمال ريسك تطبيقي است، البته با صرفنظر از آزمايش خاص ٥ (كه رخداد آن زياد هم محتمل نيست!) كه در آن به دليل خرابي تعداد زيادي از عاملها از جمله عاملهاي بسيار مهم و نياز شديد به كمك، بهترين كارايي با انتخاب احتمال زياد ريسك حاصل مي‌شود.

1 Low Risk
2 High Risk
3 Medium Risk
4 Adaptive Risk

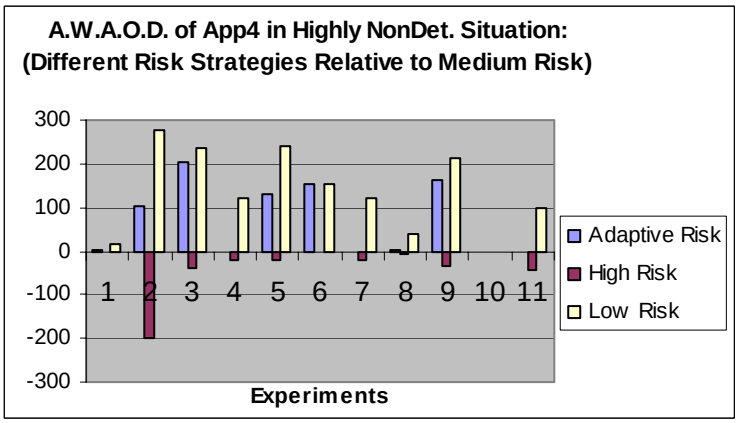
در ضمن مشاهده مي‌شود كه ريسك نمودن زياد براي كمك‌رساني، عاملها را از انجام به موقع كار خود باز مي‌دارد و موجب افزايش قابل‌توجه زمان پاسخ مي‌گردد. هرچه عامل بيشتر پايبند به انجام به موقع وظيفة خودش باشد، چنانكه با مي‌نيمم ريسك در شكل43 ديده مي‌شود، زمان پاسخ بهتري حاصل مي‌شود. مناسبترين فاصله زماني تكميل وظيفه تا مهلت زماني مقرر نيز آنچنان كه در شكل44 ديده مي‌شود، با انجام كمترين ريسك بدست مي‌آيد و با انجام ريسك زياد، عامل صرفا با تاخير در انجام وظيفة خودش مواجه مي‌گردد. علاوه بر اينها با داشتن كمترين ريسك، طبيعي است كه عامل كمترين كمك‌ها و در نتيجه كمترين ناسازگاري با كمك‌ها را بدست مي‌آورد و بنابراين در شكل45 مينيمم مقدار متعلق به كمترين ريسك است ولي با توجه به ديگر معيارها قابل قبول‌ترين مقدار متعلق به ريسك تطبيقي است.



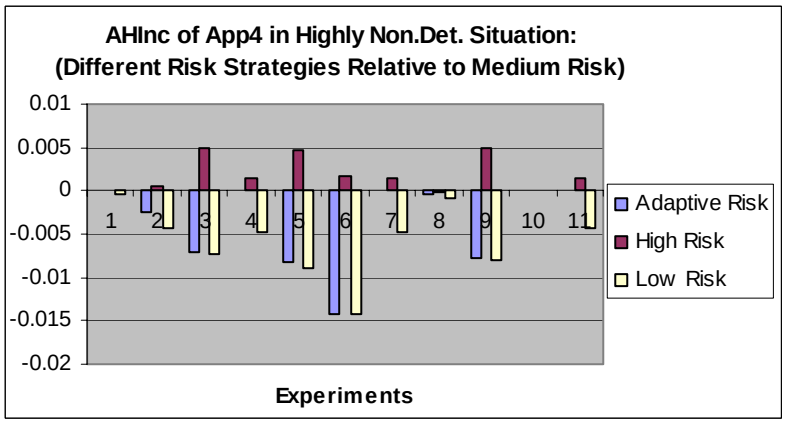
شكل 42. كارايي روش چهارم در حالت عدم قطعيت زياد در سيستم



شکل43. زمان پاسخ روش چهارم در حالت عدم قطعیت زیاد در سیستم



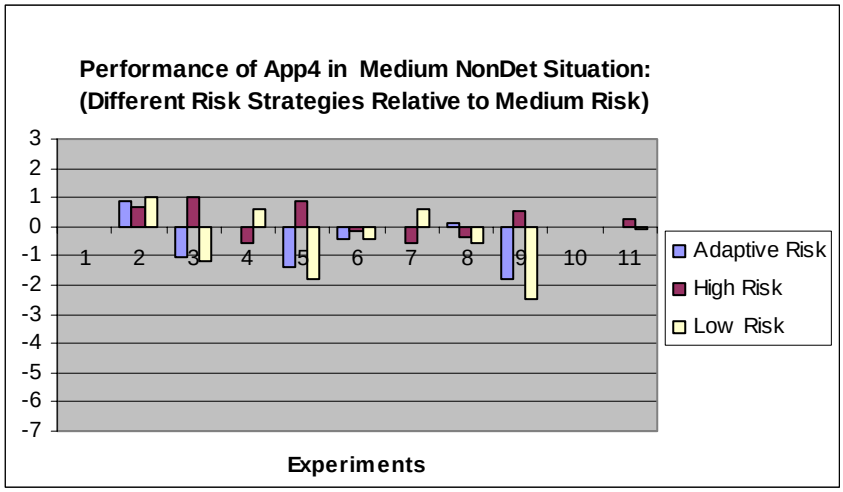
شکل 44. فاصله زماني تكميل تا مهلت مقرر روش براي چهارم در حالت عدم قطعیت زیاد در سیستم



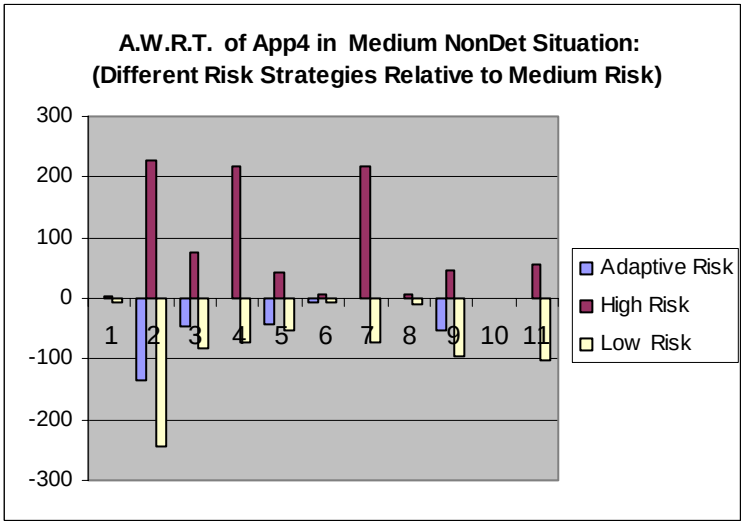
شکل 45.عدم هماهنگي كمك و كمك‌رسان در روش چهارم در حالت عدم قطعیت زیاد در سیستم

۷-۳-۲-۱-۲- بررسي عملكرد روش چهارم با عدم قطعیت متوسط در سیستم

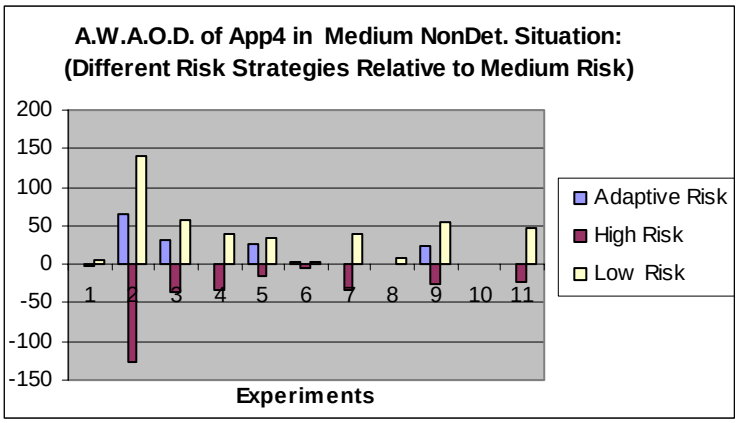
در اينجا نمودارهاي ارزيابي كارايي سيستم مبتني بر روش چهارم را با وجود عدم قطعيت متوسط و با ٤ سطح مختلف از احتمال ريسك براي عاملهاي كمك‌رسان مشاهده مي‌كنيد. توجه به نمودارهاي ارزيابي كه درشكل46 تا شكل49 ديده مي‌شوند، همانند حالت عدم قطعيت زياد، با استدلال‌هاي مشابهي تاييد مي‌كنند كه از ميان اين چهار مقدار متفاوت براي احتمال ريسك، مناسبترين آنها احتمال ريسك تطبيقي است. البته به دليل كاهش عدم قطعيت در سيستم بر خلاف حالت قبل، تفاوت عمده‌اي ميان اين مقادير از لحاظ كارايي وجود ندارد.



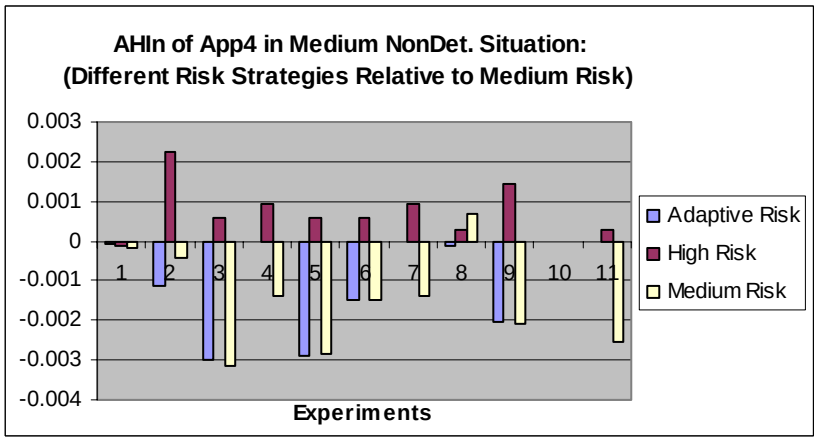
شكل 46. كارايي در روش چهارم در حالت عدم قطعيت متوسط در سيستم



شكل 47. زمان پاسخ در روش چهارم در حالت عدم قطعيت متوسط در سيستم



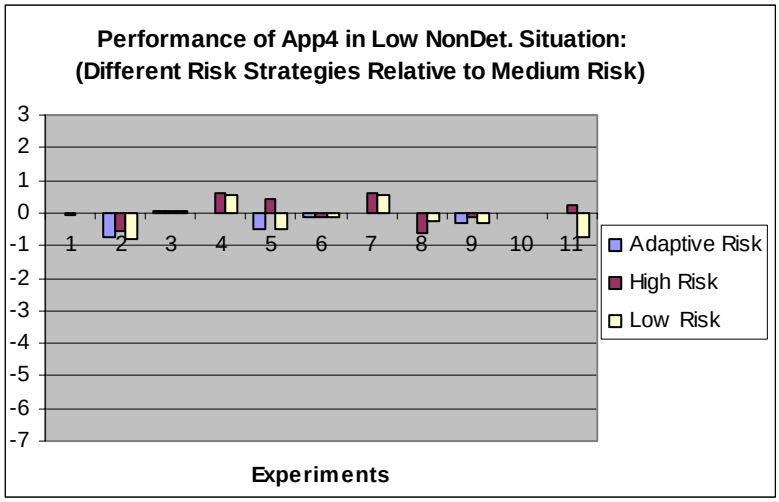
شکل 48. فاصله زماني تكميل تا مهلت مقرر در روش چهارم در حالت عدم قطعيت متوسط در سيستم



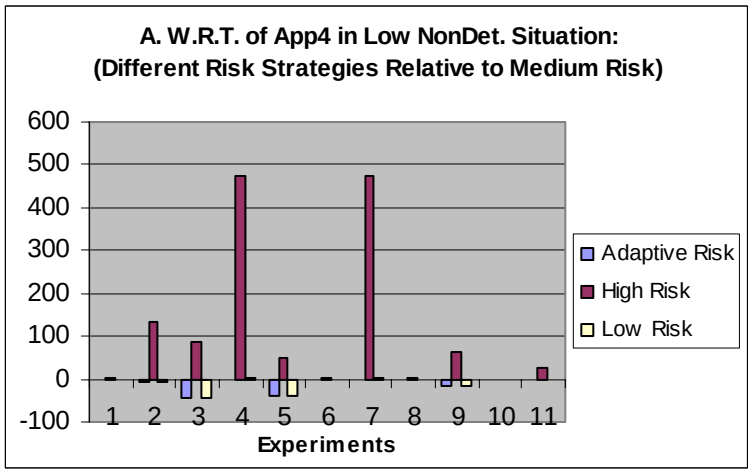
شكل49. عدم هماهنگي كمك و كمك‌رسان در روش چهارم در حالت عدم قطعيت متوسط در سيستم

۷-۳-۲-۱-۳- بررسي عملكرد روش چهارم با عدم قطعيت كم در سيستم

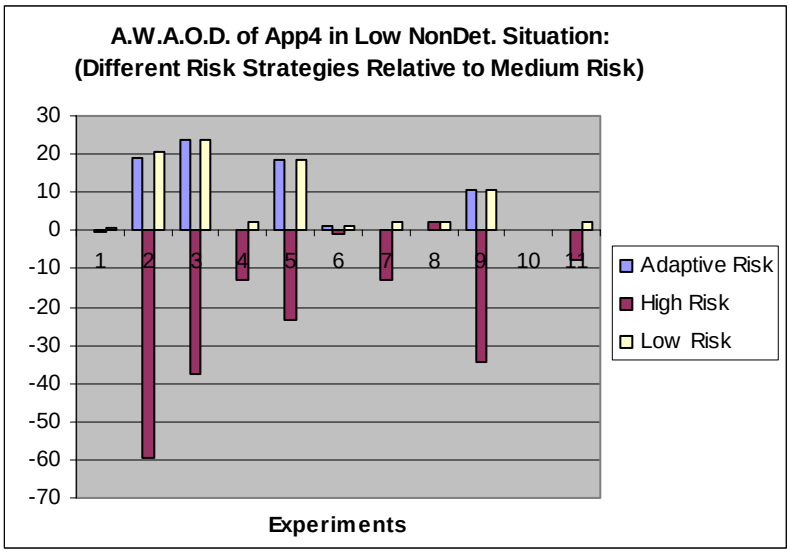
در اينجا نمودارهاي ارزيابي كارايي سيستم مبتني بر روش چهارم را با وجود عدم قطعيت كم و با ٤ سطح مختلف از احتمال ريسك براي عامل‌هاي كمك‌رسان مشاهده مي‌كنيد. توجه به نمودارهاي ارزيابي كه درشكل50 تا شكل53 ديده مي‌شوند، نشان مي‌دهند كه در صورت وجود عدم قطعيت كم در سيستم، تفاوتي ميان روشها با مقادير مختلف از احتمال ريسك وجود ندارد و تنها به دليل زمان پاسخ مناسبتر است كه از ميان اين چهار مقدار متفاوت براي احتمال ريسك، مناسبترين آنها احتمال ريسك تطبيقي خواهد بود.



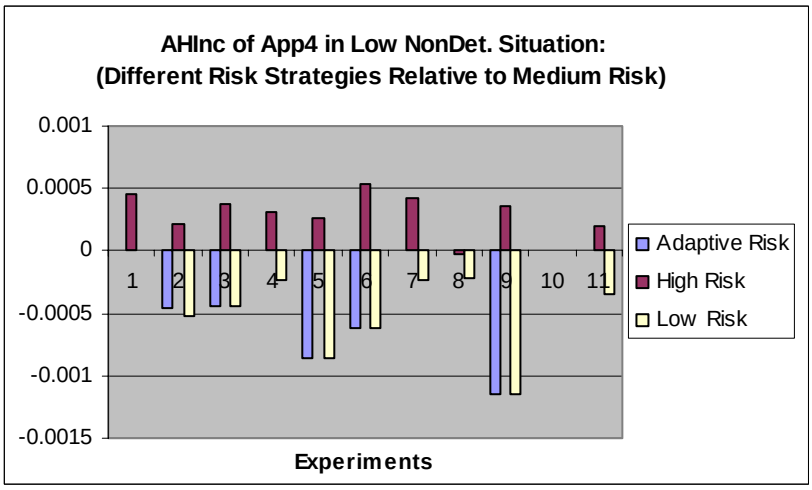
شکل 50. کارایی در روش چهارم در حالت عدم قطعیت کم در سیستم



شکل51. زمان پاسخ متوسط در روش چهارم در حالت عدم قطعیت کم در سیستم



شکل 52. فاصلة زماني تكميل وظيفه تا مهلت زماني مقرر در روش چهارم در حالت عدم قطعيت كم در سيستم



شکل 53. عدم هماهنگي ميان كمك و كمك‌رسان در روش چهارم در حالت عدم قطعيت كم در سيستم

۷-۳-۲-۲- روش پنجم

اين روش همانطور كه در بخش قبل اشاره شد، ويرايش تكميلي روش چهارم است. يعني از تقسيم بهينة وظيفه‌ها براي تعيين كمك‌رسانهاي مناسب استفاده مي‌كند، و پيش از اجراي هر وظيفه امكان‌سنجي غيرقطعي در مورد آن وظيفه با مقدار تطبيقي براي ريسك انجام مي‌دهد و با توجه به ميزان آستانة تعيين شده براي اضطراب او در زمان طراحي، تصميم مي‌گيرد كه كمك فعلي را تكميل نمايد يا به انجام كار خود اقدام نمايد.

در اين روش چهار حالت مختلف براي آستانة اضطراب عامل در نظر گرفته شده است كه اين ٤ مقدار عبارتند از:

١) آستانة اضطراب مينيمم[1]: در حد ٠
٢) آستانة اضطراب ماكزيمم[2]: در حد ١
٣) آستانة اضطراب = ٠,١
٤) آستانة اضطراب = ٠,٥

براي ارزيابي روش پنجم هم به عنوان يك سيستم غيرقطعي از هر سه حالت غيرقطعي كه براي باس ورودي سيستم متصور است نيز استفاده شده است تا عملكرد سيستم با افزايش عدم قطعيت هم مورد بررسي قرار بگيرد. نمودارهاي اين بخش نيز همگي به صورت نسبي ترسيم شده اند، به اين شكل كه معيار مقايسة خود را عملكرد سيستم با آستانة اضطراب ماكزيمم قرار داده ايم (كه در آن عامل با رسيدن كار خودش، وظيفة فعلي را رها نميكند، چرا كه همواره اضطراب عامل بر انجام كار خودش از ماكزيمم آستانه كمتر است و به اجراي وظيفة قبلي تا تكميل ادامه ميدهد) و بقية مقادير آستانه اي ديگر را در مقايسه با اين آستانه ترسيم كرده ايم.

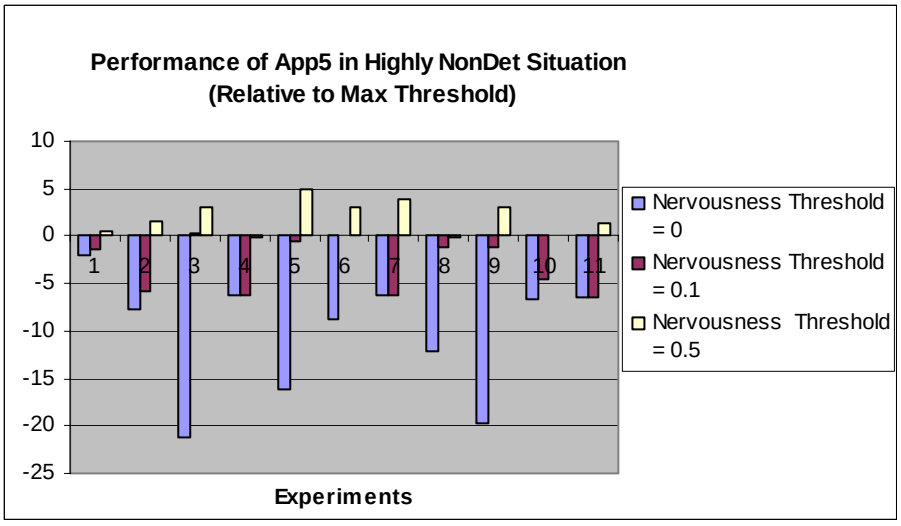
٧-٣-٢-٢-١- بررسي عملكرد روش پنجم با عدم قطعيت زياد در سيستم

در اينجا نمودارهاي ارزيابي كارايي سيستم مبتني بر روش پنجم را با وجود عدم قطعيت زياد و با ٤ سطح مختلف از آستانة اضطراب براي عاملهاي كمكرسان مشاهد ميكنيد. توجه به نمودارهاي ارزيابي كه درشكل54 تا شكل57 ديده ميشوند، تاييد ميكند كه از ميان اين چهار مقدار متفاوت براي آستانة اضطراب، در اغلب موارد و با توجه به معيارهاي اصلي ترماننـد كارايي و زمان پاسخ، مناسبترين آستانه براي اضطراب عامل ٠,٥ است، يعني اگر عامل بيش از اين مقدار براي كار خودش مضطرب باشد، رها كردن كار فعلي و پرداختن به كار خودش نه تنها به نفع خودش بلكه به نفع كل تيم خواهد بود.

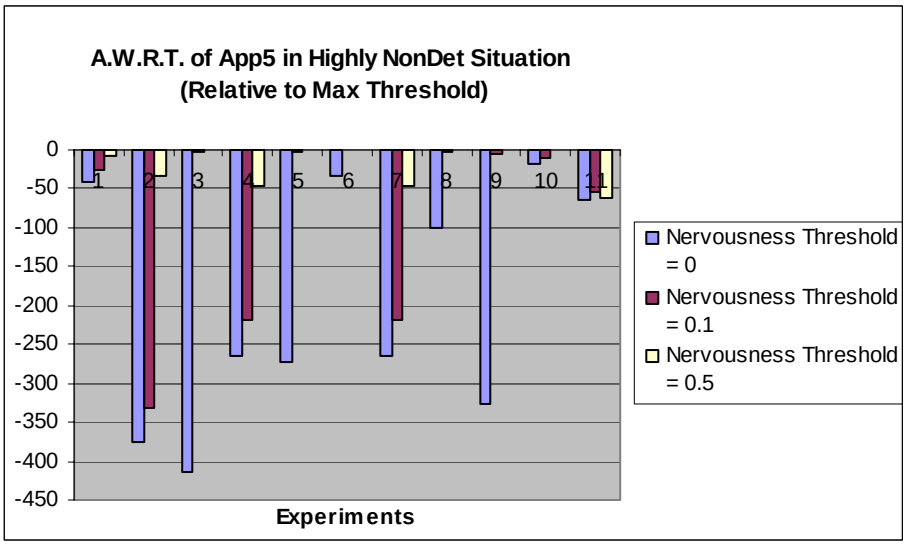
در ضمن مشاهده ميشود كه آستانة اضطراب مينيمم موجب ميگردد كه عامل همواره با رسيدن كار خودش، كار كمك فعلي كه در ميانة انجام آن است را رها نمايد و كار

1 Minimum Nervousness Threshold
2 Maximum Nervousness Threshold

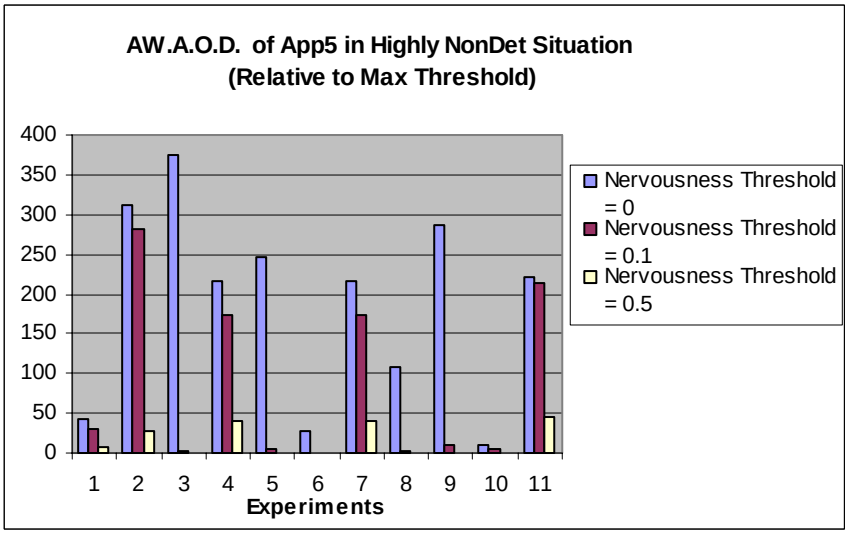
خودش را شروع کند، بدون اینکه بررسي کند که آیا واقعا نیاز به اینکار وجود دارد یا اینکه کار خودش مي‌تواند منتظر بماند تا کار فعلي او تکمیل گردد بدون اینکه مهلت زماني مقرر آن از دست برود. بنابراین داشتن آستانة اضطراب مي‌نیمم، هیچ سودي جز مي‌نیمم نمودن زمان پاسخ ندارد. البته توجه به این نکته ضروري است که در این روشها به دلیل وجود اثر مجازي یادگیري، همواره اگر کار یك عامل توسط خودش انجام بگیرد، مي‌نیمم زمان پاسخ و ماکزیمم فاصله زماني تکمیل تا موعد مقرر حاصل مي‌گردد، چرا که پس از گذشت مدتي و کسب خبرگي، هیچ کدام از عاملهاي دیگر توانایي خود او را در انجام وظیفه‌اش ندارند. اما در نظر گرفتن آستانة اضطراب ماکزیمم نیز کار چندان منطقي نیست چرا که اگر عامل نگران انجام کار خودش نباشد و همواره به امید دیگران از انجام کار خودش امتناع کند (چنانکه در روش ٤ انجام مي‌شد)،در صورتي که عامل داراي قابلیت ویژه‌اي باشد و کار خود او تنها از خودش ساخته باشد و نه عامل دیگري، بهتر آن است که عامل پیش از اتخاذ هر تصمیمي بررسي کند که انجام کدام کار بهتر است و آنگاه یا خود به انجام کارش اقدام نماید و براي تکمیل کمك از دیگران کمك بخواهد، یا اینکه براي کار خودش از بقیه درخواست کمك نماید و به تکمیل کار فعلي‌اش ادامه دهد. بنابراین ایجاد تعادلي میان این دو مساله لازم خواهد بود که به همین دلیل مقدارهاي میاني (به جز حدود ماکزیمم و مي‌نیمم) مورد بررسي قرار گرفته‌اند.



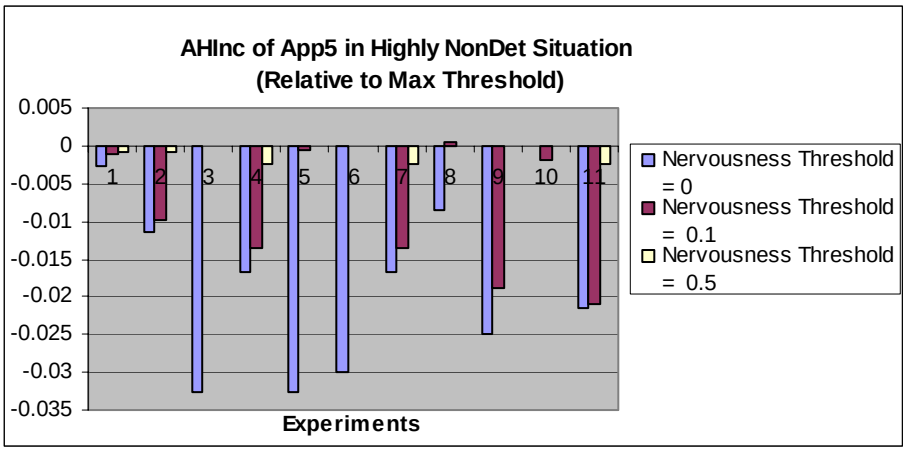
شکل 54.کارایي در روش پنجم با وجود عدم قطعیت زیاد در سیستم



شکل 55. زمان پاسخ در روش پنجم با وجود عدم قطعیت زیاد در سیستم



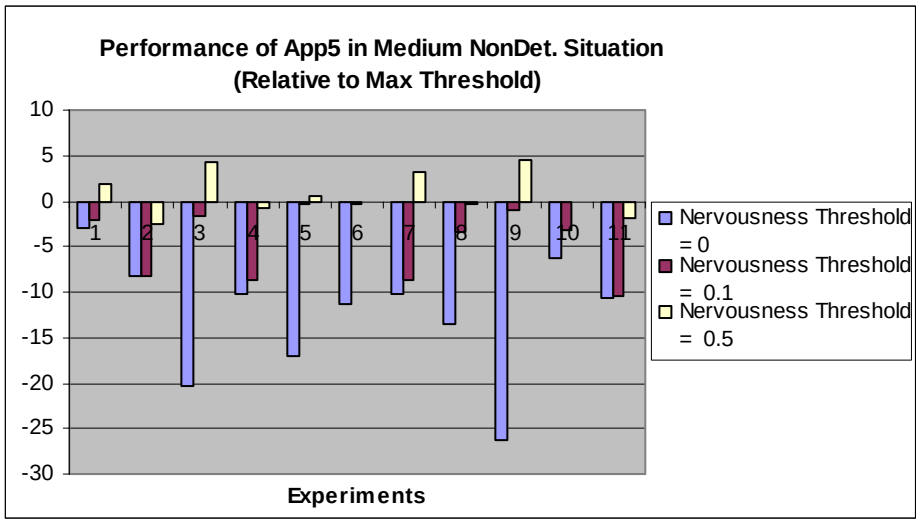
شکل 56. فاصله زماني تكميل تا مهلت زماني مقرر در روش پنجم با وجود عدم قطعیت زیاد در سیستم



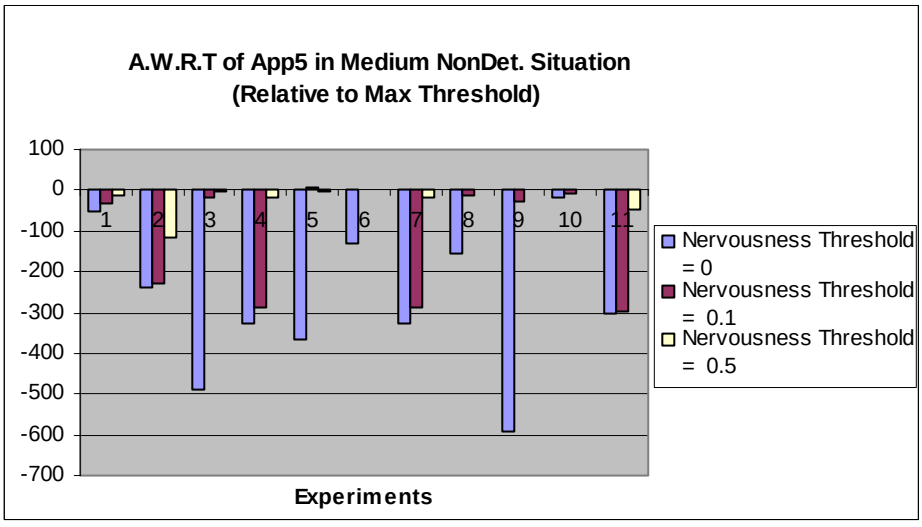
شکل 57. عدم سازگاري کمك و کمك‌رسان در روش پنجم با وجود عدم قطعیت زیاد در سیستم

۷-۳-۲-۲-۲- بررسي عملکرد روش پنجم با عدم قطعیت متوسط در سیستم

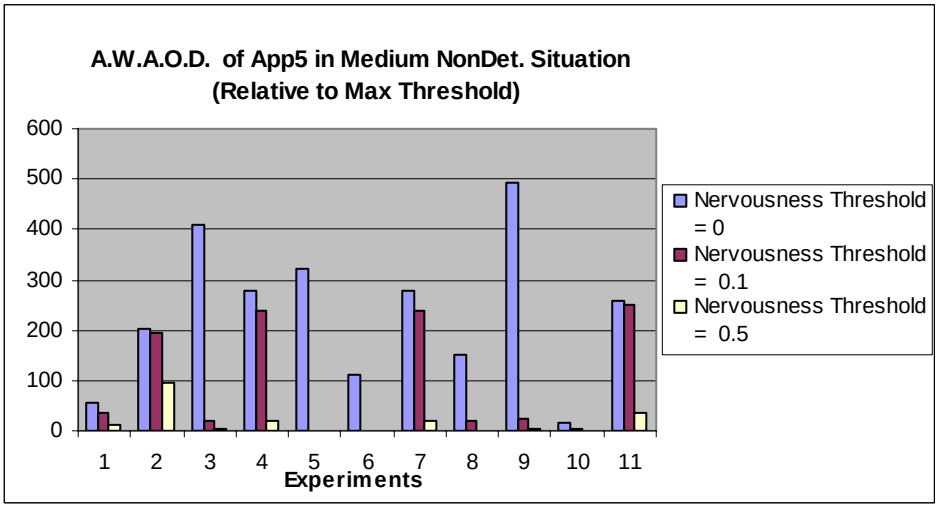
در اینجا نمودارهاي ارزیابي کارایي سیستم مبتني بر روش پنجم را با وجود عدم قطعیت متوسط و با ٤ سطح مختلف از آستانة اضطراب براي عامل‌هاي کمك‌رسان مشاهد مي‌کنید. توجه به نمودارهاي ارزیابي که درشکل58 تا شکل61 دیده مي‌شوند و ارائة استدلال‌هاي مشابه، تایید مي‌کند که از میان این چهار مقدار متفاوت براي آستانة اضطراب، مناسبترین آنها آستانة اضطراب ۰,۵ است، هر چند که این مقدار، معیارهاي فاصله زماني تکمیل تا مهلت مقرر و نیز عدم هماهنگي میان کمك و کمك‌رسان را به صورت بهینه به‌دست نمي‌دهد، ولي با توجه به اینکه دو معیار اول یعني کارایي و زمان پاسخ اهمیت بیشتري در ارزیابي دارند، این مقدار براي آستانه انتخاب مي‌گردد. هرچند که با کاهش عدم قطعیت حساسیت این انتخاب هم از بین مي‌رود و چنانکه در بخش بعدي مشاهده مي‌کنید با عدم قطعیت کم، تفاوتي میان انتخاب این مقادیر وجود نخواهد داشت.



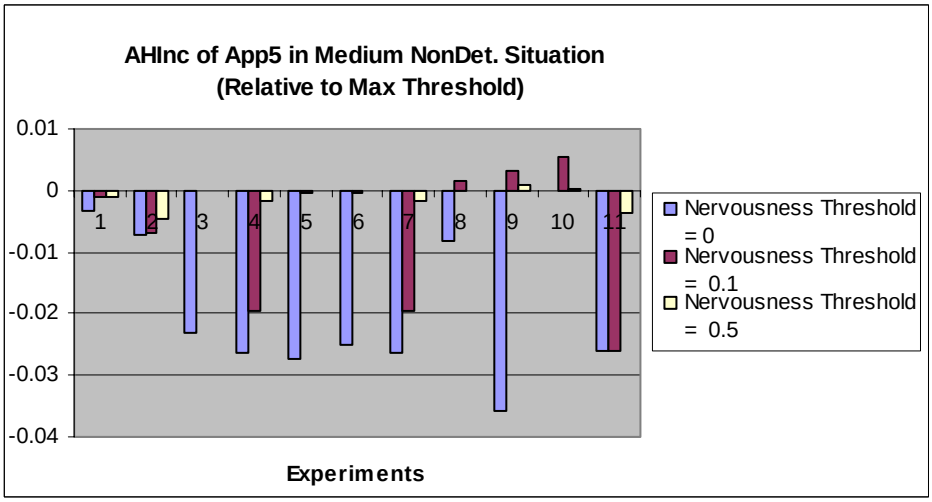
شکل 58. کارایی در روش پنجم با وجود عدم قطعیت متوسط در سیستم



شکل 59. پاسخ زمانی در روش پنجم با وجود عدم قطعیت متوسط در سیستم



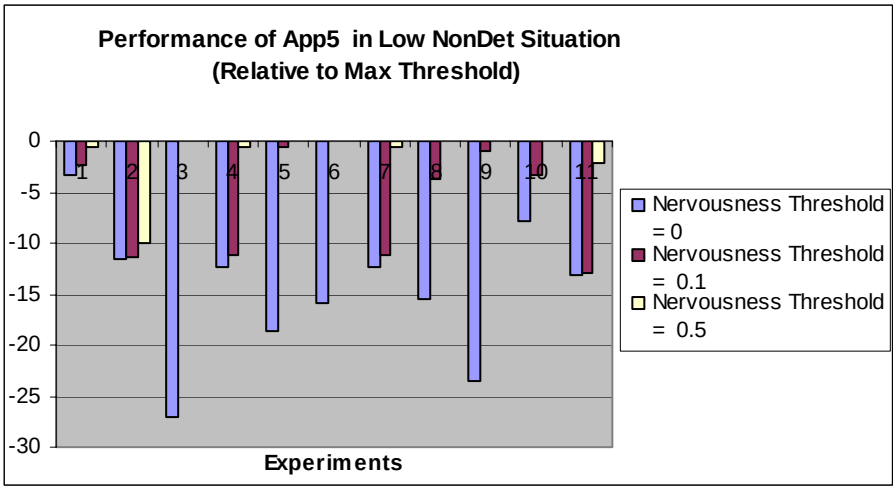
شکل60. . فاصلة زماني تكميل تا مهلت مقرر در روش پنجم با وجود عدم قطعيت متوسط در سيستم



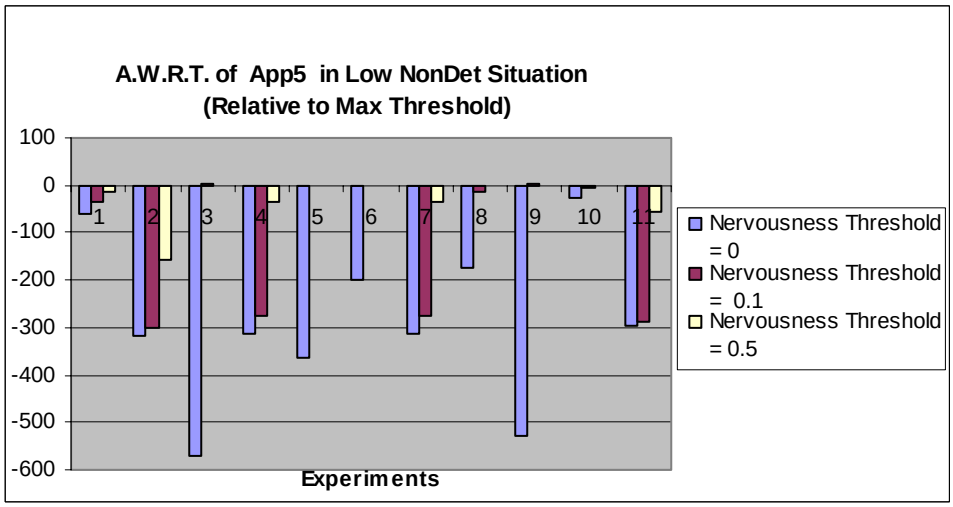
شكل 61. عدم هماهنگي كمك و كمك‌رسان در روش پنجم با وجود عدم قطعيت متوسط در سيستم

۷-۳-۲-۲-۳- بررسي عملكرد روش پنجم با عدم قطعيت كم در سيستم

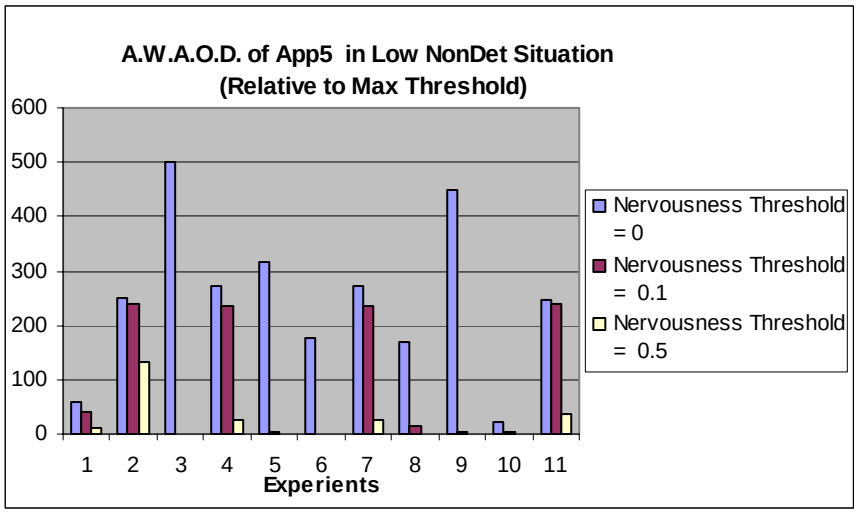
در اينجا نمودارهاي ارزيابي كارايي سيستم مبتني بر روش پنجم را با وجود عدم قطعيت كم و با ٤ سطح مختلف از آستانة اضطراب براي عامل‌هاي كمك‌رسان مشاهد مي‌كنيد. توجه به نمودارهاي ارزيابي كه در شكل62 تا شكل65 ديده مي‌شوند و ارائة استدلال‌هاي مشابه، تاييد مي‌كند كه از ميان اين چهار مقدار متفاوت براي آستانة اضطراب، همواره مناسبترين آنها آستانة اضطراب ۰,۵ است.



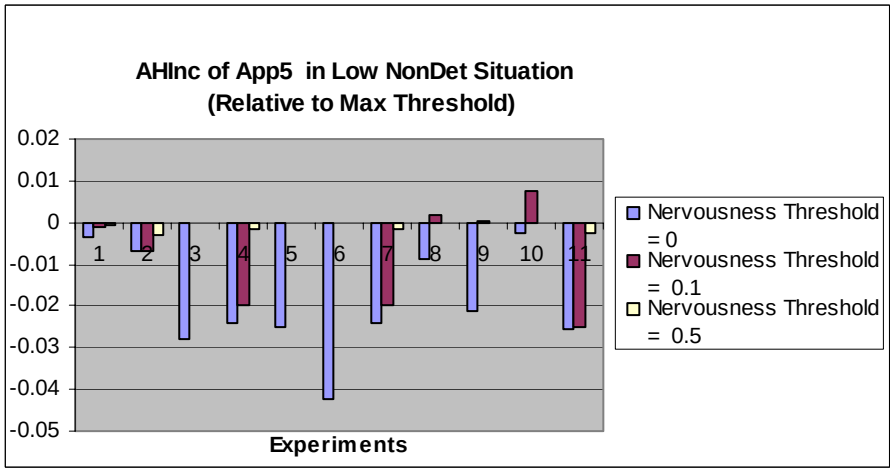
شکل 62. کارایی در روش پنجم با وجود عدم قطعیت کم در سیستم



شکل63. زمان پاسخ در روش پنجم با وجود عدم قطعیت کم در سیستم



شکل 64. فاصله زماني تكميل وظيفه تا مهلت زماني مقرر در روش پنجم با وجود عدم قطعيت كم در سيستم



شکل 65. عدم هماهنگي كمك و كمك‌رساني در روش پنجم با وجود عدم قطعيت كم در سيستم

۷-۳-۲-۳- مقايسة روشهاي غيرقطعي با يكديگر و روش ۳ در حالت اعمال ورودي غيرقطعي

در اينجا براي مقايسة روشهاي پيشنهادي بايكديگر و براي يكسان شدن شرايط آزمايش، به روش سوم كه يك روش قطعي است، داده هايي با پريود غيرقطعي اعمال شد و عملكرد سيستم مورد بررسي قرار گرفت. در نمودارهاي زير عملكرد سيستم در روش چهارم را با احتمال ريسك تطبيقي با عملكرد سيستم در روش پنجم، با آستانة اضطراب ۰,۵ (كه به عنوان بهترين مقادير انتخابي به دست آمدند) و نيز روش سوم با هم مورد مقايسه قرار گرفته اند و نتايج در نمودارهاي زير در ۳ حالت عدم قطعيت (زياد، متوسط و كم) نشان داده شده است. لازم به توضيح است كه

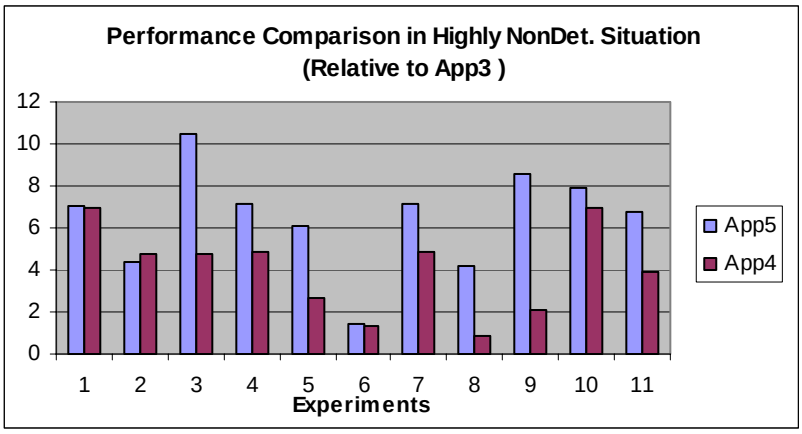
به منظور بهبود وضوح نمودارها، نمايش نسبي عملكرد نسبت به روش سوم انتخاب شده است.

۷-۳-۲-۳-۱- مقايسة روشها در حالت عدم قطعيت زياد در سيستم

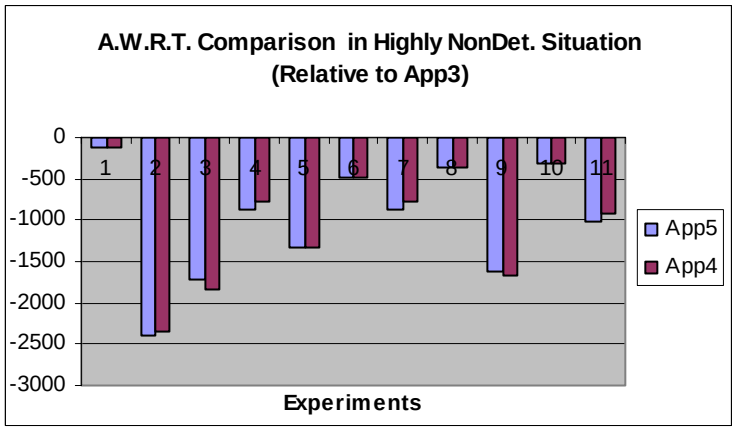
در اين قسمت به ارزيابي روشها در حالتي كه داده‌هايي با پريود بسيار غيرقطعي به سيستم داده شده است، مي‌پردازيم. نمودارهاي مربوطه كه درشكل66 تا شكل69 نشان داده شده‌اند، تاييد مي‌كنند كه در صورتي كه در سيستم عدم قطعيت قابل توجهي موجود باشد، بهترين روش از لحاظ كارايي، زمان پاسخ و فاصله زماني تكميل تا مهلت زماني مقرر روش پنجم است. در عين حال مشاهده مي‌شود كه اين روش سازگاري خوبي هم ميان كمك و كمك‌رسان ايجاد نموده است.

همانطور كه از شرح روشها در فصل قبل به ياد داريد، روش پنجم به دليل رها نكردن وظيفة عامل كمك‌رسان براي ديگران و تصميم‌گيري منطقي در قبال آن، از كارايي مناسبتري نسبت به روش چهارم برخوردار بود.

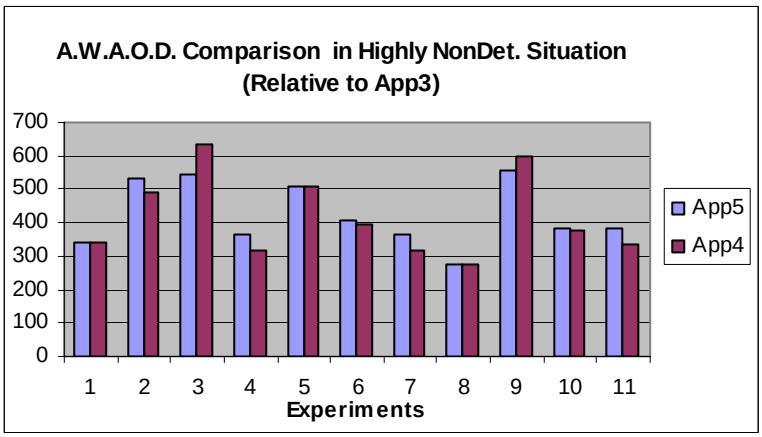
با توجه به شكل67 و شكل68 مشاهده مي‌شود كه روش سوم به دليل بي‌اطلاعي از نايقيني موجود در سيستم همواره با قطعيت صددرصد اقدام به كمك‌رساني نموده است و اين موجب شده كه انجام وظيفه‌هاي خود عامل‌ها كه مهمترين عناصر موجود در فرمول ارزيابي زمان پاسخ هستند، با تاخير انتظار قابل‌توجهي مواجه شوند و بدين ترتيب زمان پاسخ افزايش قابل‌ملاحظه‌اي بيابد و زمان تكميل وظيفه به مهلت مقرر نزديكتر گردد. علاوه بر اينها آنچنان كه شكل69 نشان مي‌دهد، روش سوم به دليل كمك‌هايي كه بدون توجه به مسالة عدم قطعيت انجام داده‌است و طبيعتا به دليل از دست رفتن كار خود كمك‌رسان علي‌رغم انتظار موجب افزايش كارايي هم نشده است، عدم سازگاري بيشتري ميان كمك و كمك‌رسان را موجب گرديده است.



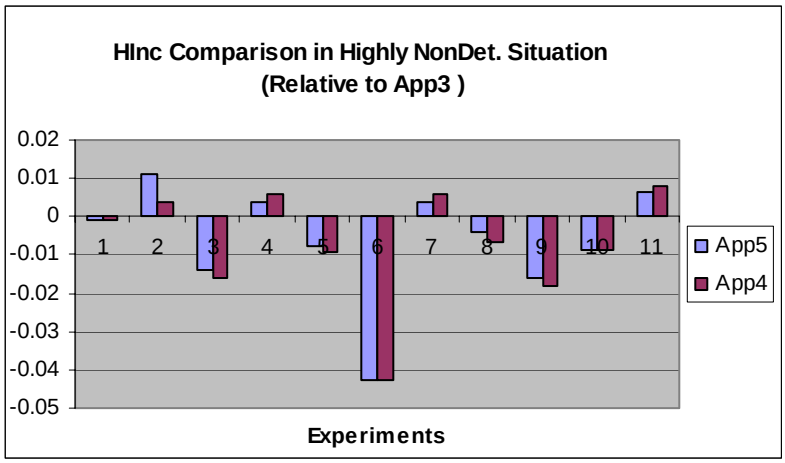
شکل66. مقایسة کارایي روشها در حالت عدم قطعیت زیاد در سیستم



شکل 67. مقایسة زمان پاسخ روشها در حالت عدم قطعیت زیاد در سیستم



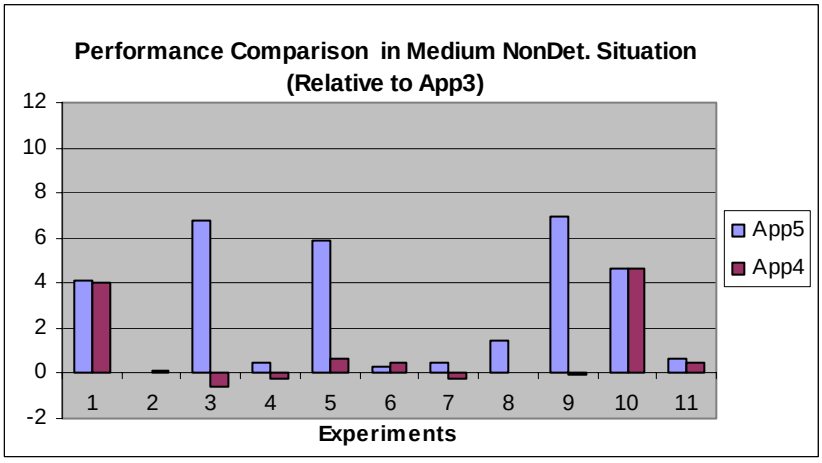
شکل 68. مقایسة فاصله زماني تکمیل تا مهلت زماني مقرردر روشها در حالت عدم قطعیت زیاد در سیستم



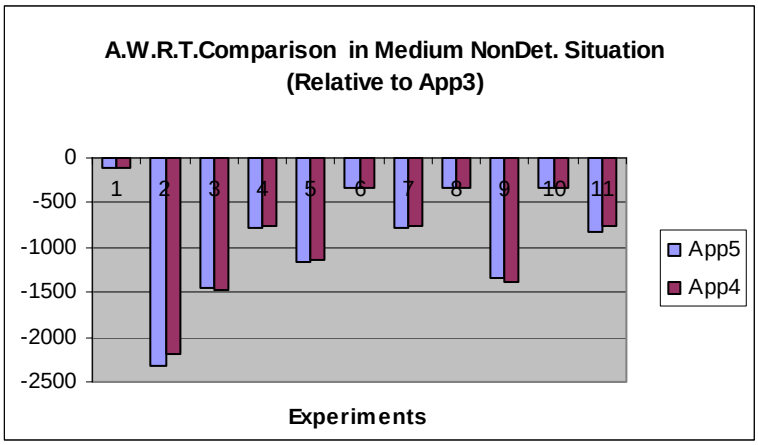
شكل 69. مقايسة عدم سازگاري كمك و كمك‌رسان در روشها در حالت عدم قطعيت زياد در سيستم

۷-۳-۲-۳-۲-مقايسة روشها در حالت عدم قطعيت متوسط در سيستم

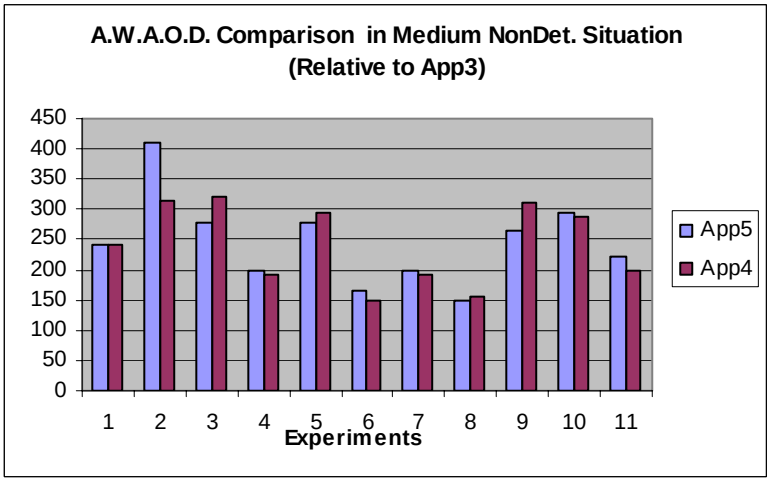
در اين قسمت به ارزيابي روشها در حالتي كه داده‌هايي با پريود نسبتا غيرقطعي به سيستم داده شده است، مي‌پردازيم. نمودارهاي مربوطه كه درشكل70 تا شكل73 نشان داده شده‌اند، تاييد مي‌كنند كه با كاهش عدم قطعيت در سيستم، عملكرد روشهاي قطعي قابل قبول‌تر مي‌شود ولي بازهم مشابه قبل و با ارائة استدلالهاي مشابه مي‌توان به اين نتيجه رسيد كه بهترين روش از لحاظ كارايي، زمان پاسخ و فاصله زماني تكميل تا مهلت زماني مقرر روش پنجم است.



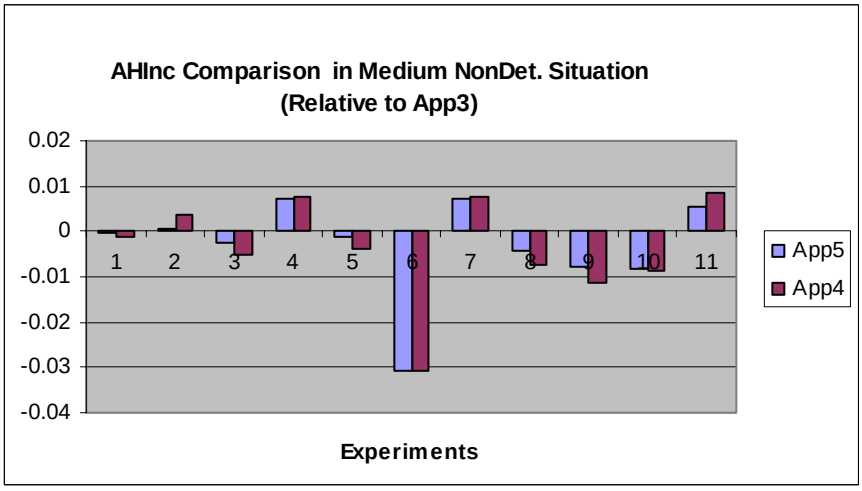
شكل 70. مقايسة كارايي روشها در حالت عدم قطعيت متوسط در سيستم



شکل71. مقایسة زمان پاسخ روشها در حالت عدم قطعیت متوسط در سیستم



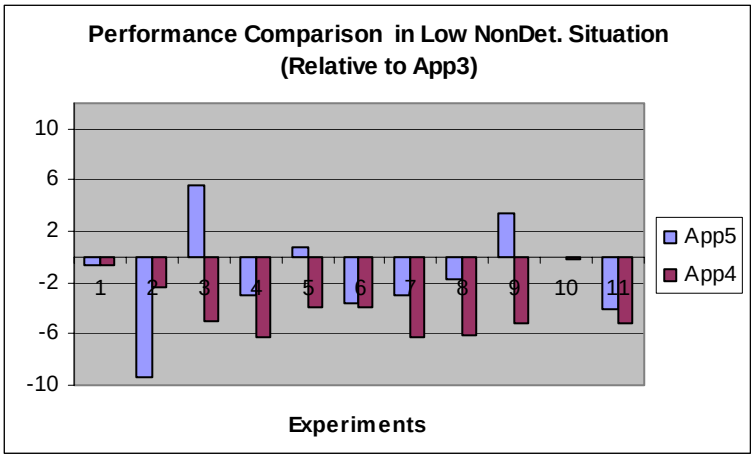
شکل72. مقایسة فاصله زماني تكميل تا مهلت زماني مقرردر روشها در حالت عدم قطعیت متوسط در سیستم



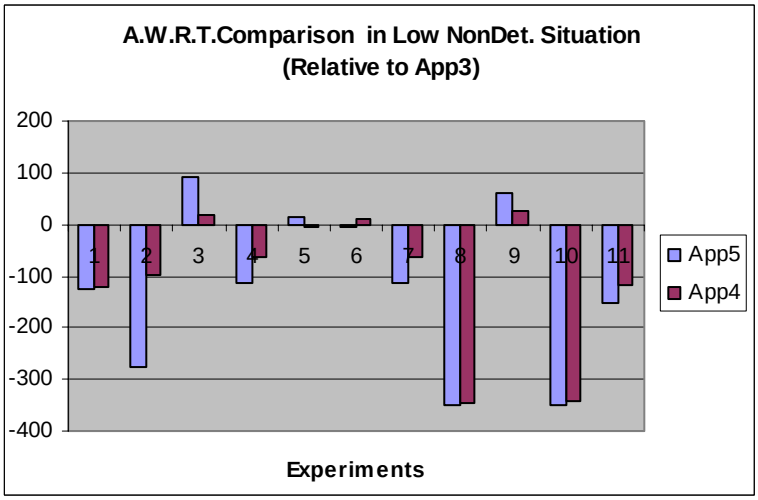
شکل73. مقایسة عدم سازگاري كمك و كمك‌رسان در روشها در حالت عدم قطعیت متوسط در سیستم

۷-۳-۲-۳-۳- مقایسة روشها در حالت عدم قطعیت کم در سیستم

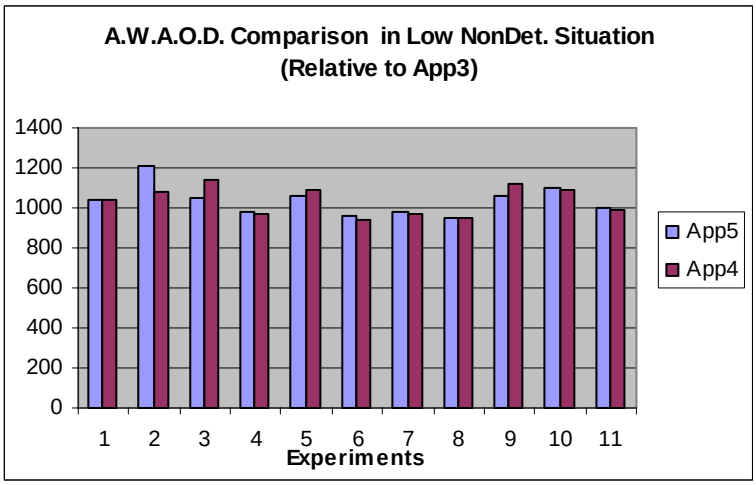
در این قسمت به ارزیابي روشها در حالتي كه داده‌هايي با پريود نسبتا قطعي به سيستم داده شده است، مي‌پردازيم. نمودارهاي مربوطه كه در شكل74 تا شكل77 نشان داده شده‌اند، تاييد مي‌كنند كه با كاهش عدم قطعيت در سيستم، عملكرد روشهاي قطعي قابل قبول‌تر مي‌شود و شايد با صرفنظر نمودن از مسالة كيفيت زماني پاسخ و اطمينان از پايين بودن سطح عدم قطعيت از يك روش قطعي مانند روش ۳ در سيستم استفاده نمود.



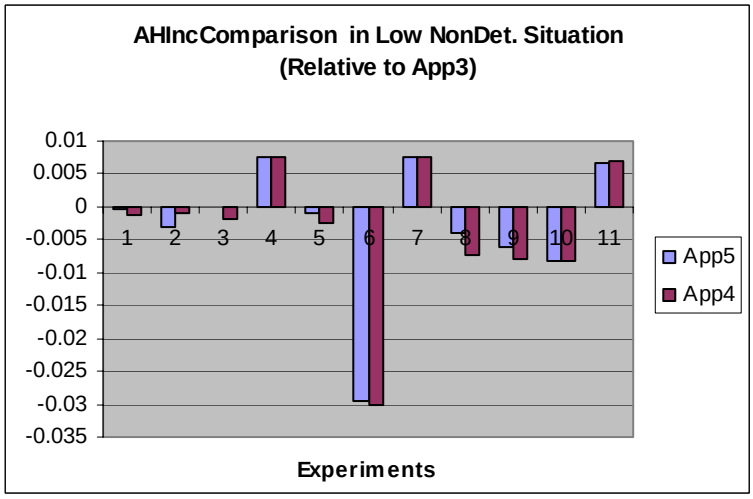
شكل74. مقايسة كارايي روشها در حالت عدم قطعيت كم در سيستم



شكل 75. مقايسة زمان پاسخ روشها در حالت عدم قطعيت كم در سيستم



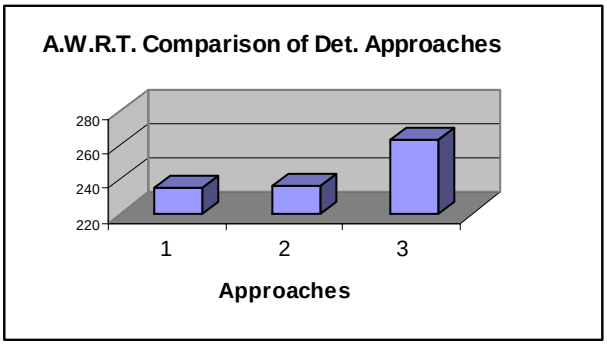
شکل 76. مقایسة فاصله زماني تکمیل تا مهلت زماني مقرردر روشها در حالت عدم قطعیت کم در سیستم



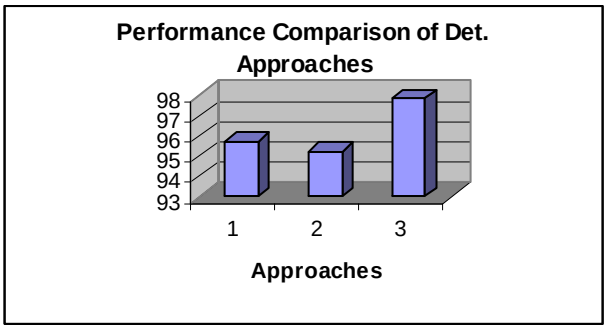
شکل 77. . مقایسة عدم سازگاري کمك و کمك‌رسان در روشها در حالت عدم قطعیت کم در سیستم

۷-٤- خلاصه

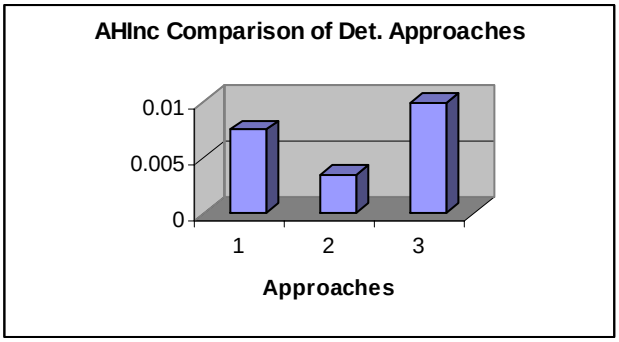
به عنوان خلاصه‌اي از نتایج آزمایشها، از متوسط وزندار نتایج بالا با احتساب احتمال رخداد هر یك از الگوهاي خرابي استفاده کرده‌ایم. به عنوان مثال احتمال به وقوع پیوستن الگویي که در آن دو عامل با ضرایب قابلیت‌اعتماد ۲ و ۳ دچار خرابي مي‌شود، متناسب است با $\frac{1}{6}$. به همین ترتیب با در نظر گرفتن قابلیت اطمینان عاملهاي خراب در هر الگو، احتمال رخداد آن الگو را محاسبه نموده، آن را در محاسبة متوسط کلي وزندار وارد کردیم که نتایج مقایسة روشهاي قطعي در نمودارهاي شکل78 تا شکل83 به قرار زیر است:



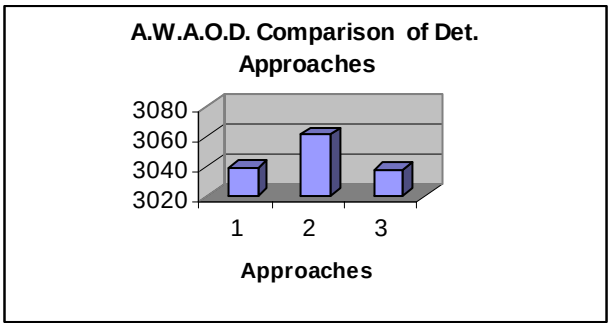
شكل 78. مقايسة زمان پاسخ متوسط وزن‌دار سه روش قطعي



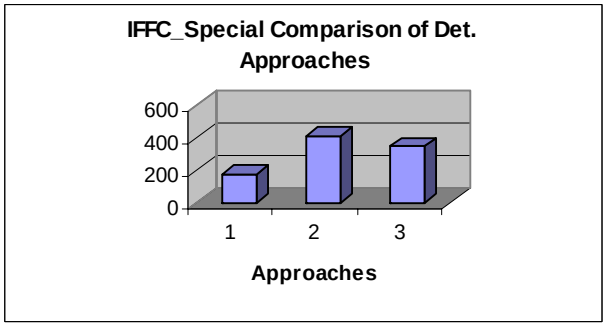
شكل79. مقايسة كارايي سه روش قطعي



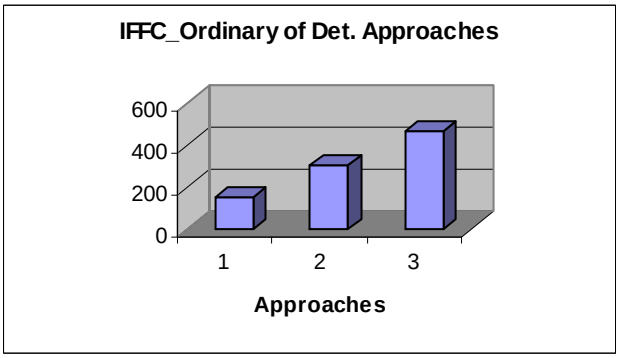
شكل 80. مقايسة ناهماهنگي كمك و كمك‌رسان در سه روش قطعي



شكل81. مقايسة فاصله زماني تكميل تا مهلت مقرر در سه روش قطعي

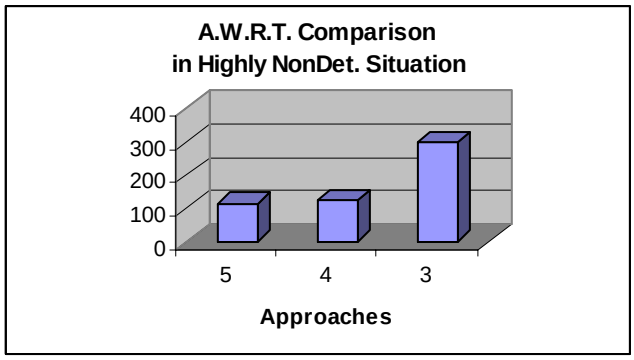


شكل82. مقايسة ميزان كنارآمدن با خرابي‌هاي ناگهاني آتي به كمك عاملهاي معمولي رزرو

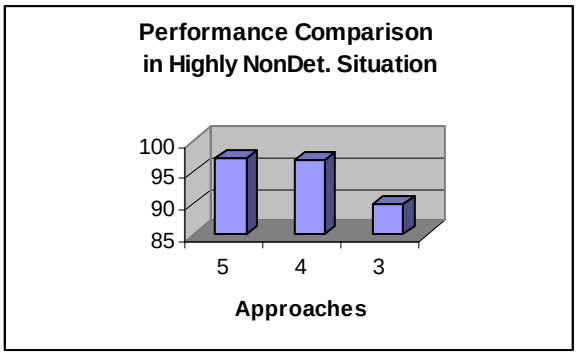


شكل83. مقايسة ميزان كنارآمدن با خرابي‌هاي ناگهاني آتي به كمك عاملهاي ويژة رزرو

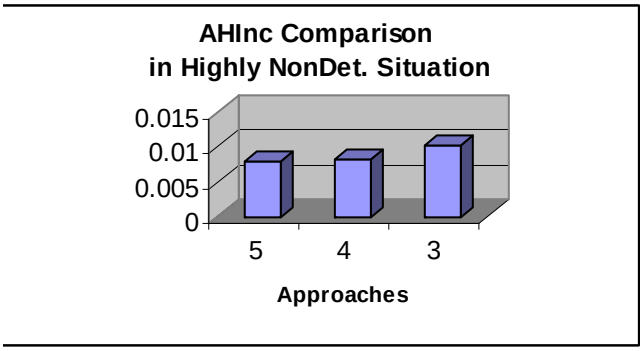
مقايسة روشهاي غير قطعي و روش سوم (با اعمال دادة غيرقطعي) نيز به شرح زير در شكل84 تا شكل95 آورده شده است:



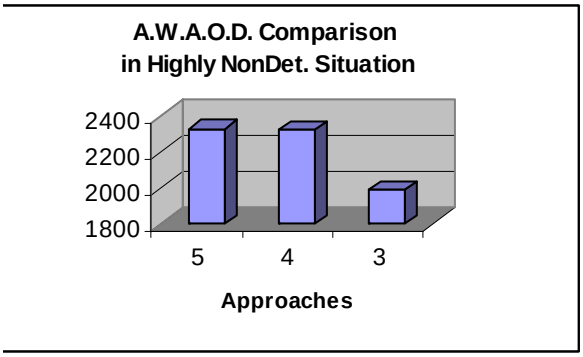
شکل 84. مقایسة زمان پاسخ متوسط وزندار روشهاي غيرقطعي در حالت عدم قطعيت زياد



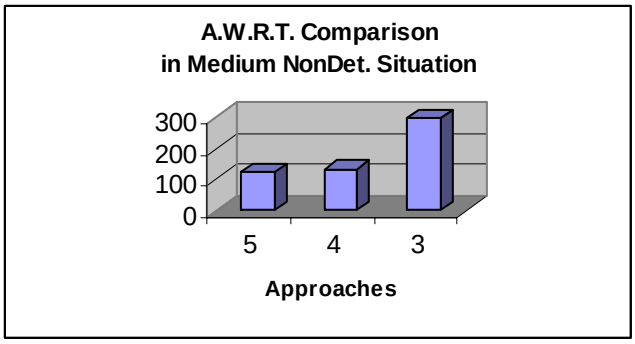
شکل 85. مقایسة کارایي روشهاي غيرقطعي در حالت عدم قطعيت زياد



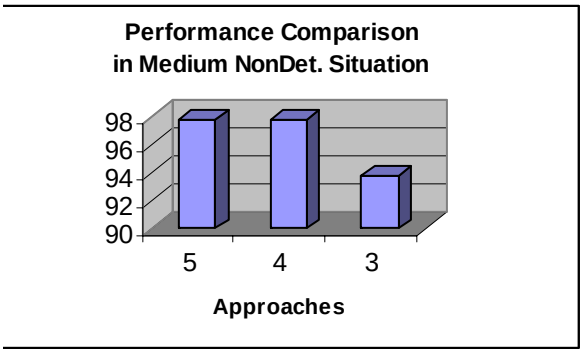
شکل 86. مقایسة ناهماهنگي کمك و کمك‌رسان در روشهاي غيرقطعي در حالت عدم قطعيت زياد



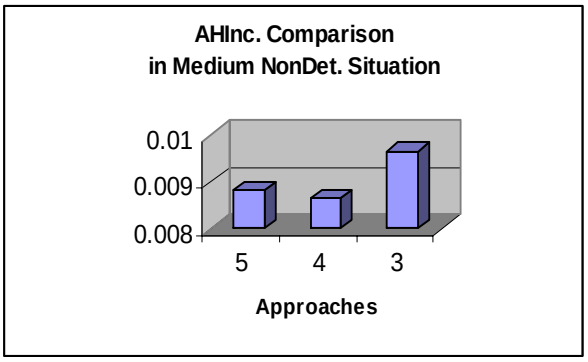
شکل87. مقایسة فاصله زماني تکميل تا مهلت مقرر در روشهاي غيرقطعي در حالت عدم قطعيت زياد



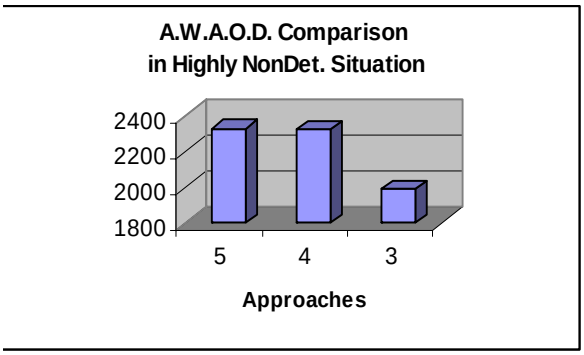
شکل88. مقایسة زمان پاسخ متوسط وزندار روشهاي غيرقطعي در حالت عدم قطعيت متوسط



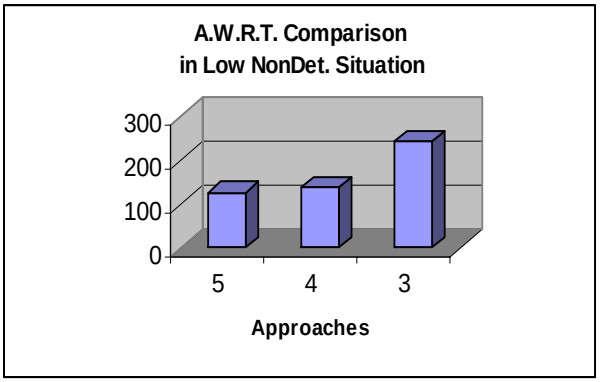
شکل89. مقایسة کارایي روشهاي غيرقطعي در حالت عدم قطعيت متوسط



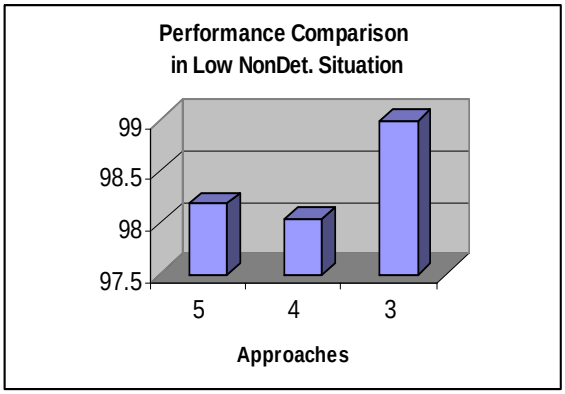
شکل 90 . مقایسة ناهماهنگي کمك و کمك‌رسان در روشهاي غيرقطعي در حالت عدم قطعيت متوسط



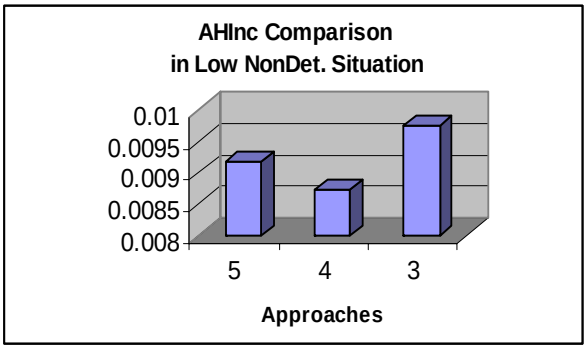
شکل 91. مقایسة فاصله زماني تکميل تا مهلت مقرر در روشهاي غيرقطعي در حالت عدم قطعيت متوسط



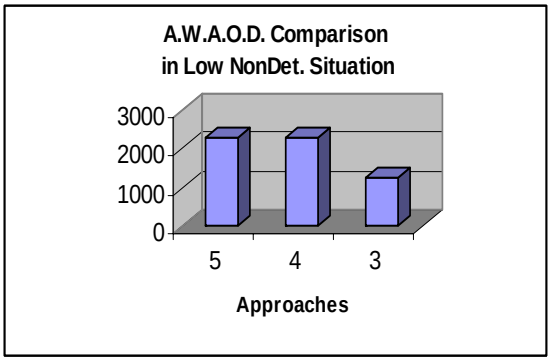
شکل 92 مقایسة زمان پاسخ متوسط وزندار روشهاي غيرقطعي در حالت عدم قطعيت کم



شکل93. مقایسة کارايي روشهاي غيرقطعي در حالت عدم قطعيت کم



شکل 94. مقایسة ناهماهنگي کمك و کمك‌رسان در روشهاي غيرقطعي در حالت عدم قطعيت کم



شکل95. مقایسة فاصله زماني تکميل تا مهلت مقرر در روشهاي غيرقطعي در حالت عدم قطعيت کم

۸- سنتز و نتايج شبيه‌سازي پس از سنتز

در اين فصل مختصرا به سنتز طرح اول كه طرح ساده اي است ولي در عين حال چنانكه در فصل آزمايشها ونتايج بيان شد، داراي كارايي قابل قبولي است، اشاره خواهد شد. البته از ميان دو سياست موجود براي روش اول، سياست SJF-over-FPF را كه از كارايي كلي بهتري برخوردار است، استفاده شده است.

۸-۱- مقدمه

علت انتخاب طرح اول براي انجام عمليات سنتز، پيچيدگي كم اين طرح است كه در مقايسه با طرح سوم كه داراي پيچيدگي هاي الگوريتميك مي باشد، از سادگي نسبي برخوردار است.

به عنوان يك يادآوري، طرح پيشنهادي اول داراي ماجولهايVHDL زير است :

1_Include.vhd
2_FrameGenerator.vhd
3_CommunicationChannel.vhd
4_Agent.vhd
5_RSFF.vhd
6_FaultGenerator.vhd
7_Multi Agent System.vhd

از بين كلية اين موجوديت[1]ها، 4_Agent.vhd براي تبديل به طرح قابل سنتز انتخاب شده است، چرا كه در اصل كل ايده هاي اين تحقيق و عمدة توجه اين پروژه معطوف به طراحي عاملهايي است كه داراي قابليت تحمل پذيري خطا باشند و ديگر مولفه ها براي تحقق يك بستر آزمون براي ارزيابي ايده هاي ما، طراحي شده اند.

۸-۲- تغييرات لازم براي تبديل به مدل قابل سنتز

براي قابل سنتز نمودن اين طرح، تغييراتي در كد توصيف رفتاري VHDL هر عامل اعمال شده است.

تفاوتهاي عمدة طرح قابل سنتز با طرح اوليه عمدتا در حذف Timingهايي است كه در نمونة شبيه سازي شده براي واقعي تر جلوه كردن عملكرد وجود داشت (از قبيل در نظر گرفتن

1 Entity

تاخير در اجراي وظيفة عاملها، تاخير در تكميل كمك رساني و...) ولي در طرح قابل سنتز اين تاخيرها به دليل انجام عمليات واقعي سخت افزاري درنظر گرفته نشده است. تفاوت ديگر، استفاده از سرعتهاي نسبي متفاوتي براي عاملها نسبت به حالت پيش از سنتز است، به اين دليل كه، قابل سنتز بودن عمليات تقسيم مستلزم آن است كه در فرمول زير حتما مقسوم عليه توانی از ۲ باشد تا با عمليات شيفت به راست پياده سازي گردد.

ComputationTime = *AmountOfWork / CPUSpeed*

به علاوه به دليل غير قابل سنتزبودن انواع سيگنالهاي فيزيكي از جملة زمان، و به علت استفاده از زمان رسيد درخواستهاي كمك و نيز زمان رسيد وظيفه براي عاملها در ايده هاي طراحي روشهاي اين تحقيق، مجبور به استفاده از يكClock مجازي با مقدار Integer شديم كه در هر لحظه، زمان كنوني را نه از جنس زمان بلكه از نوع Integer در اختيار ما بگذارد. در اصل ورودي Clock با دورة تناوب ۱ نانوثانيه به مولفة عامل، نقص اطلاعاتي زماني موجود در طرح قابل سنتز را برطرف نمود.

بنابراين، براي جايگزيني NOW با يك معادل قابل سنتز، از مقدار سيگنال Integer با نام CLK_1NS استفاده كرديم.

مورد بعدي، تغيير جملاتي بود كه حاوي تاخير بودند و حتما لازم بود كه به نحوي كه قابل سنتز باشد، تغيير يابند. براي مثال يكي از تغييراتي كه در طرح قابل سنتز لازم بود كه داده شود، جايگزين نمودن عباراتي نظير عبارات زير با جملات قابل سنتز است:

```
Process (…)
 Begin
  If condition then
    A<= ‘1’, ‘0’ AFTER 1 NS;
  End if
End Process;
```

براي قابل سنتز نمودن چنين عباراتي، پروسسي شبيه پروسس زير لازم است كه در آن، يك سيگنال CLK به سيگنالهاي موجود در Sensitivity List اضافه شده باشد.

```
Process(…, Clk_1NS)
Begin
```

```
A <= '0';
If condition then
  A <= '1';
End if;
End Process;
```

طبيعتا يك نانوثانيه پس از ۱ شدن سيگنال A، با وقوع يك رخداد بر روي Clk_1NS، مجددا وارد پروسس مي شويم و A خودبه خود، صفر خواهد شد.

تغيير لازم ديگر اينكه، چون ورودي و خروجي عاملها كه از روي باس ورودي برداشته يا روي باس خروجي نوشته مي شود، از نوع ركورد با فيلدهاي Encode شده هستند، ولي در طرح پس از سنتز، همة انواع معتبر داده اي به نوع Std-logic تك بيتي تبديل مي شوند، يك Decoder و يك Encoder براي تبديل انواع داده اي لازم خواهد بود.

۸-۳- جزئيات عمليات سنتز

با اين تفاسير، اين طرح به كمك Leonardo Spectrum 2000 و با استفاده از FPGA از نوع Altera – Flex10K سنتز شد. سپس فايل VHDL به همراه توابع كتابخانه اي Flex10K براي شبيه سازي پس از سنتز به محيط شبيه سازي يعني ModelSim انتقال يافت و در كنار ديگر مولفه هاي سيستم كه سنتز نشده بودند، مورد شبيه سازي قرار گرفت.

از آنجا كه براي اين مدار با كاربردي كه براي آن متصور است، اولويت با بهينه سازي بر اساس سرعت است و نه مساحت، پس از بهينه‌سازي بر اساس سرعت انجام گرفت و نتايج به قرار زير است:

- ✓ تعداد Logic Cell ها : ۱۱۰٤۱
- ✓ تعداد TriState Buffer ها : ۸
- ✓ تعداد D-Flip Flop ها : ٤۸۸
- ✓ تعداد پورتهاي ورودي و خروجي به ترتيب : ۷۳ و ۳۹.
- ✓ تاخير مسير بحراني: ۴۱۹ نانوثانيه

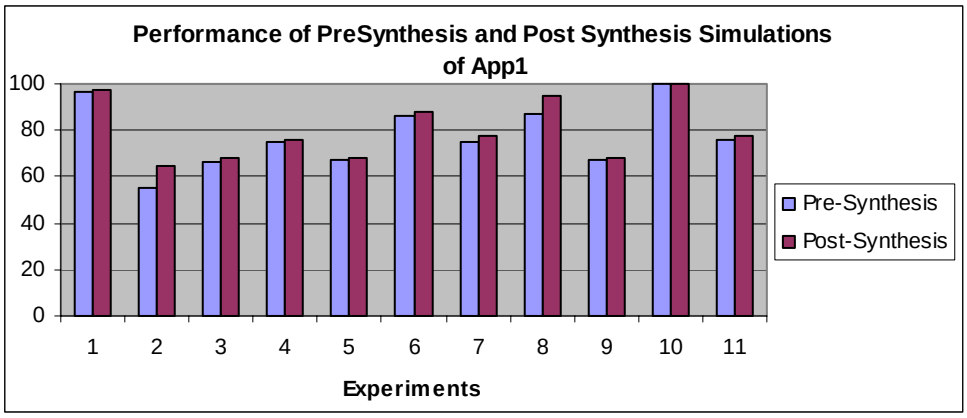
در سيستم چندعاملة مورد آزمون، هر عامل را بامدل حاصل از سنتز آن جايگزين نموده‌ايم. نتايج شبيه سازي پس از سنتز در بخش بعد آورده شده است.

۸-٤- نتايج شبيه سازي پس از سنتز

در اينجا نتايج اعمال ۳۰ الگوي خرابي كه در فصل قبل معرفي شده‌اند به سيستم چندعاملة طراحي شده در اين تحقيق را نشان مي‌دهيم. لازم به ذكر است كه از ميان تمامي معيارهاي ارزيابي معرفي شده در فصل قبل، تنها كارايي مورد ارزيابي قرار گرفته است. به اين دليل كه در مدل قابل سنتز، نوشتن اطلاعات موردنياز براي محاسبة فرمولهاي ارزيابي در فايل امكان‌پذير نبوده و ارزيابي دقيق اين معيار براي ۳۰ آزمايش و ۱۰ عامل در هر آزمايش، از روي شكل موج كار بسيار دشواري است.

در اصل در اين حالت هم عاملها همان الگوريتم تصميم‌گيري قبلي را براي انتخاب مناسبترين كمك‌رسان اجرا مي‌كنند و بنابراين در انتخاب كمك‌رسانها و ترتيب آنها، هيچ تفاوتي وجود ندارد. تنها تفاوت اين است كه زمان آزادي عاملها كه مي‌توانند آن را صرف كمك‌رساني نمايند از حالت قبلي بسيار بيشتر است، چرا كه انجام وظيفه از آنها زماني نمي‌گيرد و بنابراين به كلية عاملهايي كه براي كمك‌رساني انتخاب كرده‌اند، كمك مي‌كنند و با كمبود زمان مواجه نمي‌شوند.

در شكل96، كارايي روش اول با سياست SJF-over-FPF در دو حالت پيش و پس از سنتز مورد مقايسه قرار گرفته‌اند. مشاهده مي‌شود كه در آزمايشهاي ساده‌تر دو سيستم شباهت بسياري به هم دارند، اما در آزمايشهايي كه سخت‌تر هستند و نياز به كمك عاملها به شدت احساس مي‌شود و عاملها به دليل كمبود زمان خالي كه مي‌توانند صرف كمك‌رساني نمايند از كمك‌رساني بازمانده‌اند (و نه به دليل ضعف در تقسيم وظيفه‌ها، چرا كه ضعف در تقسيم مناسب وظيفه‌ها در روش اول همچنان وجود دارد)، به دليل سرعت بيشتر در تكميل وظايف (كه دليل آن لحاظ نكردن تاخير انجام وظيفه هاست)، قادر به كمك‌رساني شده‌اند و اين بر كارايي افزوده است. . البته در اين حالت، همانطور كه گفته شد، عاملهاي با سرعت نسبي ۳ نيز به دليل محدوديت عمليات سنتز با سرعت ٤ در نظر گرفته شدند كه اينهم بر سرعت انجام وظيفة آنها مي‌افزايد. مواردي مانند آزمايش ۲ و ۸ كه در آنها تعداد خرابي‌ها بسيار زياد بوده است، اين مساله را به خوبي نشان مي‌دهند.



شکل 96. مقايسة كارايي دو سيستم در حالت پيش و پس از سنتز

۹- نتيجه‌گيري و كارهاي آينده

در اين تحقيق، روش‌هايي براي ايجاد تحمل‌پذيري خرابي در سيستم‌هاي گسترده به ويژه سيستم‌هاي چندعامله ارائه شدكه تفاوت اصلي اين روشها با آنچه قبلا و در اغلب پژوهشهاي مرتبط انجام شده است، متكي نبودن بر افزونگي صرف و در عوض استفاده از قابليتهاي اضافه‌تر هر عامل موجود در طرح است به شكلي كه عاملها در صورت بروز خرابي، با همكاري با يكديگر و تعويض نقش خود از حالت عادي مسئوليتهاي بيشتري را بر دوش بگيرند.

در طرح‌هاي پيشنهادي، افزونگي نه به صورت صريح و متمركز به شكل عامل‌هاي افزونه، بلكه به صورت توزيع شده و ضمني در قابليت عاملهاي اصلي سيستم قرار مي‌گيرد تا با استفاده از اين قابليت ذخيره شده، عاملها بتوانند در صورت بروز خرابي احتمالي براي هر يك از هم‌تيمي‌هاي خود، وظيفة يك يا تعدادي از آنها را نيز (بسته به ميزان قابليت‌هاي خودش) بر دوش گيرد و از كاهش كارايي سيستم تا زمان برطرف شدن خرابي، جلوگيري نمايد.

بنابراين وجود افزونگي ضمني در زمان تقسيم اولية وظيفه ميان عاملها، در صورتي كه در زمان بروز خطا به نحو شايسته‌اي به كار گرفته شود، موجب كاهش هزينة زمان طراحي سيستم در مقايسه با سيستمي مي‌شود كه براي هر يك از عناصر خود، از يك يا دو عنصر افزونه استفاده نموده است.

علاوه بر اين، استفاده از طرح‌هاي پيشنهادي، معيارهاي ارزيابي معرفي شده را تا حد قابل قبولي ارضا مي‌نمايد و اين حاكي از مفيدبودن اين ايده‌ها در سيستمهايي با شرايط مشابه است. به عنوان مقايسه‌اي اجمالي از هر يك از روشهاي ارائه شده، مي‌توان موارد زير را يادآور شد:

روش اول : تقسيم وظيفه بر اساس سرعت ذاتي - بدون ملاحظه از نظر سازگاري كمك و كمك‌رسان و با تصميم‌گيري قطعي[1]

اين روش داراي مي‌نيمم پيچيدگي زمان طراحي و محاسباتي در زمان اجراست و به دليل مبتني بودن تقسيم وظيفه ميان عاملها بر سرعت اجرايي ذاتي عاملها، هماهنگي

1 Speed-Based Task Distribution with Deterministic Decision-Making And No Help-Helper Compatibility

ميان قابليتهاي عامل كمك‌رسان و نيازهاي عاملي كه به او كمك مي‌شود وجود ندارد. به علاوه به مسالة مهم *آزادگذاشتن عاملهاي سالم براي كمك‌رساني فوري و ضروري در آينده* نيز در اين طرح توجهي نشده است.

روش دوم : تقسيم وظيفه بر اساس سرعت ذاتي - اعمال ملاحظاتي از نظر سازگاري كمك و كمك‌رسان با تصميم‌گيري قطعي[1]

اين روش كه بهبود يافتة طرح اول از لحاظ هماهنگي ميان عاملهاي كمك‌رسان و عاملهاي نيازمند كمك است، طبيعتا داراي پيچيدگي اندكي بيشتر در طراحي بوده اما همچنان براي تقسيم وظيفه متكي بر سرعت اجرايي ذاتي عاملهاست. در اين طرح عاملها در صورتي كه ديگري را براي كمك‌رساني مناسبتر از خود تشخيص بدهند، از كمك صرفنظر مي‌كنند. اين روش از لحاظ كارايي اغلب از روش اول ضعيفتر است، اما به دليل ملاحظاتي كه در طراحي آن اعمال شده است، از لحاظ سازگاري كمك و كمك‌رسان و همچنين ذخيرة عاملهاي كمك‌رسان به صورت رزرو براي خرابي‌هاي احتمالي آينده، منطقي‌تر عمل مي‌نمايد.

روش سوم: تقسيم وظيفه بر اساس الگوريتم نسبتا بهينه با تصميم‌گيري قطعي[2]

اين طرح كه مناسبترين كارايي را در روشهاي قطعي به‌دست آورده است، مبتني بر تقسيم وظيفه ميان عاملها بر اساس الگوريتم نسبتا بهينه است، از پيچيدگي زمان اجرا و بار محاسباتي بيشتري برخوردار است. عاملهاي كمك‌رسان در زماني كه به مرحلة كمك‌رساني مي‌رسند، با اجراي اين الگوريتم، درخواستهاي كمك را به نحو مطلوبي ميان خود تقسيم مي‌كنند و اين موجب مي‌گردد كه تعداد عاملهاي كمك‌رسان آزاد كه بر اساس الگوريتم تقسيم وظيفه، هيچ كاري به آنها تخصيص نيافته است و آمادة كمك‌رساني فوري و ضروري در آينده هستند، در سيستم افزايش يابد. علاوه بر اين به دليل در نظر گرفتن مسالة هماهنگي ميان عامل كمك‌رسان و عامل نيازمند كمك در اين طرح، روش سوم از ميان روشهاي يك، دو و سه كه همگي در دستة روشهاي قطعي هستند، بهترين روش شناخته مي‌شود. تنها مشكل اين طرح،

1 Speed-Based Task Distribution with Help-Helper Compatibility and Deterministic Decision-Making
2 Task Distribution Based on Sub-Optimal Task Selection Algorithm and Deterministic Decision-Making

نپرداختن به مسالة عدم قطعيت است كه در طرحهاي ٤ و ٥ به آنها پرداخته شده است.

روش چهارم : تقسيم وظيفه بر اساس الگوريتم نسبتا بهينه با تصميم‌گيري غيرقطعي و سياست اجرا تا تكميل وظيفة فعلي حتي در صورت رسيدن وظيفة جديد[1]

اين طرح، ويرايش تكميل يافتة طرح ٣ است، با در نظر گرفتن مسالة عدم قطعيت در سيستم. بدين معنا كه وظيفه‌هاي عاملها در اين طرح توسط يك باس غيرقطعي ورودي (بر اساس توزيع گوسي با (μ,σ) مشخص) به عاملها منسوب مي‌شوند و بنابراين عاملها ديگر به صورت قطعي قادر به تعيين انجام‌پذير بودن يا نبودن يك وظيفه در زمان تصميم‌گيري به كمك نيستند و بايد پس از طي نمودن يك مرحلة تصميم‌گيري كه شامل عدم‌قطعيت است و با ريسك همراه مي‌باشد، تعيين نمايند كه آيا كمك مورد تقاضا انجام‌پذير است يا خير و سپس اقدام مناسب را انجام بدهند. البته از آنجا كه عامل در مورد زمان دقيق رسيدن وظيفة بعدي خود اطلاع قطعي ندارد، در صورتي كه اتمام كمك مورد تقاضا را پيش از رسيدن وظيفة بعدي خود محتمل بداند، اقدام به كمك مي‌كند و وظيفة خود را در صورت محول شدن ميان كار كمك‌رساني، به عهدة ديگر هم‌تيمي خود مي‌گذارد.

اين طرح از لحاظ پيچيدگي زماني علاوه بر نياز به زمان براي انجام محاسبات الگوريتم تقسيم نسبتا بهينة وظيفه، براي تصميم‌گيري در قبال انجام‌پذيربودن يا نبودن يك وظيفه بر اساس محاسبات احتمالاتي نيز مستلزم پيچيدگي‌هاي بيشتري خواهد بود كه به بهاي كارايي بهتر، كنارآمدن نايقيني‌هاي موجود و تامين نيازمندي‌هاي تحمل‌پذيري خطا، قابل‌قبول مي‌نمايد.

روش پنجم: تقسيم وظيفه بر اساس الگوريتم نسبتا بهينه با تصميم‌گيري غيرقطعي و بررسي ميزان اضطراب در صورت رسيدن وظيفة جديد[2]

1 Task Distribution Based on Sub-Optimal Task Selection Algorithm and Non-Deterministic Decision-Making with Run-to-Completion Policy even when a New Task comes in.

2 Task Distribution Based on Sub-Optimal Task Selection Algorithm and Non-Deterministic Decision-Making with Impatience Verification when New Task comes in.

اين روش، تكميل‌يافتة طرح ٤ است، به اين شكل كه در صورت رسيدن يك وظيفه براي خود عامل كمك‌رسان و مطلع‌شدن وي از رسيدن وظيفة جديد توسط ارسال يك وقفه به كار فعلي‌اش، اين عامل كه ريسك نموده و اقدام به كمك كرده، صرفا به صادر نمودن يك درخواست كمك به ديگر عاملها اكتفا نمي‌كند بلكه ميزان تمايل / اضطراب خود را براي شروع فوري كار جديد خودش يا تكميل كار قبلي كه به انجام آن مشغول بوده است، بررسي مي‌كند و در صورتي كه ميزان تمايل به شروع كار جديد را كافي بيابد، كار كمك‌رساني فعلي را رها كرده و چون قبلا آن را تقبل نموده بود، با اعلام رها نمودن آن از بقية هم‌تيمي‌ها كمك مي‌خواهد. البته در صورت كافي نبودن ميزان تمايل به شروع كار جديد،كار قبلي را تكميل مي‌كند و براي انجام كار خودش از بقيه كمك مي‌طلبد. كارايي اين روش در مقايسه با روش قبل كه آن هم به نحو ساده‌تري، قابليت كنارآمدن با نايقيني را فراهم مي‌نمود، مطلوبتر است و از لحاظ معيارهاي ارزيابي زماني مانند زمان‌پاسخ متوسط به طور محسوسي قابل قبول‌تر است ولي از لحاظ تامين تحمل‌پذيري خطا توسط عاملهاي رزرو تفاوت محسوسي ميان دو روش اخير وجود ندارد.

جدول ١ ، مقايسة كِيفي روشهاي پيشنهادي را نشان مي‌دهد:

جدول 4.مقايسة كِيفي روشهاي پيشنهادي

			خرابي‌هاي عادي		خرابي‌هاي ويژه		
		پيچيدگي محاسباتي	كارايي	زمان پاسخ	كارايي	زمان پاسخ	باقي گذاشتن عاملهاي آزاد براي آينده
شرايط قطعي	روش يك	كم	قابل قبول	مناسب	نامناسب	زياد	نامناسب
	روش دو	متوسط	قابل قبول	مناسب	نامناسب	زياد	متوسط
	روش سه	زياد	خوب	مناسب	خوب	مناسب	خوب
شرايط غيرقطعي	روش چهار	زياد	خوب	مناسب	خوب	مناسب	خوب
	روش پنج	زياد	خوب	مناسب	خوب	مناسب	خوب

به عنوان ادامة كار هريك از موارد زير مي‌تواند مدنظر باشد:

١) كنار آمدن با خرابي‌هاي جزئي[1] : در تمامي روشهاي پيشنهادي خرابي عاملها به صورت كلي فرض شده است بدين معنا كه اگر مشكلي براي عاملي

1 Partial Fault Handling

پيش بيايد فرض بر اين است كه از تمامي قابليتهايش ساقط مي‌شود و در هيچ يك از فعاليتهاي قبلي توان ادامة كار را ندارد و بنابراين كلية زيروظيفه هايش اعم از جمع‌آوري دادة خودش، انجام وظيفة حالت عادي خودش و احيانا كمك‌رساني توسط اوبه ديگر هم‌تيمي‌هايش بر زمين مي‌ماند. در حالي كه در كاربردهاي عملي غالبا تمامي قابليت‌ها از دست نمي‌رود و مي‌توان تنها در برخي زيروظيفه‌ها انتظار كمك داشت. با اضافه نمودن اين مساله به طرحهاي موجود مي‌توان بر كيفيت عملكرد از لحاظ بهينه نمودن زمان و استفادة بهتر از امكانات محاسباتي عاملها افزود.

۲) همكاري ميان عاملهاي كمك‌رسان در تكميل يك وظيفه : در همة روشهاي پيشنهادي، تنها يك عامل كمك‌رسان بايد تمام كار يك عامل خراب را تقبل نمايد و هيچ‌گونه همكاري ميان عاملها در اين مورد وجود ندارد و اين خود به نوعي موجب عدم استفادة بهينه از توان محاسباتي موجود در عاملهاي سالم در زمان خرابي در سيستم مي‌گردد. در اين صورت دو يا چند عامل مي‌توانند با يكديگر براي انجام يك وظيفه همكاري نمايند و با تقسيم آن به زيروظيفه‌هايي عملكرد مطلوبتري حاصل نمايند.

۳) انتقال مسالة تقسيم وظيفة اوليه از فاز طراحي به ابتداي زمان اجرا : در حال حاضر و در كلية روشهاي پيشنهادي، تقسيم وظيفة اوليه ميان عاملها بر اساس حجم متوسط وظيفه‌هاي موجود در سيستم، درجه اهميت تكميل به موقع اين وظيفه براي سيستم، سرعت نسبي پردازندة هر عامل و ضريب قابليت اطمينان به پردازندة هر عامل و در فاز طراحي صورت مي‌گيرد. در صورتي كه در زمان اجرا، هر عامل بتواند بر اساس قابليتهايي كه دارد،مناسبترين وظيفه را براي خود انتخاب نمايد، بر هوشمندي سيستم افزوده خواهد شد.

٤) استفاده از مكانيزمهاي بازار[1] در ايجاد انگيزة كمك براي عاملها : يك ديدگاه متفاوت براي ايجاد انگيزة كمك‌رساني براي عاملها مكانيزم مبتني بر

1 Market-based

سود و زیان موجود در روشهای بازار است که در این پروژه نیز امکان بهره‌گرفتن از آنها وجود دارد.

١٠- مـراجع

[١] Barry Johnson, "Design and Analysis of Fault Tolerant Digital Systems", Adison Wesely, 1989

[٢] Dhiraj K. Pradhan, "Fault-Tolerant Computer System Design", 1995, Prentice Hall

[٣] Levi Agrawala, "Fault Tolerant System Design", 1994, McGrawHill

[٤] Siewiorek and Swarz, "Reliable Computer Systems", 1992, Digital Press

[٥] Stothert and MacLeod, "Multi_Agent Techniques for Fault Tolerance in Computer Control Systems", Department of Electrical Engineering, University of The Witwatersrand, Johannesburg South Africa, 1997

[٦] Chrysanthos Dellarocas and Mark Klein, "An Experimental Evaluation of Domain-Independent Fault Handling Services in Open Multi-Agent Systems", ICMAS2000

[٧] Staffan Hugg, "A Sentinel Approach to Fault Handling in Multi-Agent Systems", Department of Computer Science, University of Karlskrona/Ronneby, Sweden

[٨] Sanjeev Kumar , Philip R. Cohen, "Towards a Fault-Tolerant Multi-Agent System Architecture", Oregon Graduate Institute, Autonomus Agent 2000

[٩] Sanjeev Kumar, Philip R. Cohen, Hector J. Levesque, "The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams", Oregon Graduate Institute, University of Toronto, Autonomus Agent 2000

[١٠] N.Vijaykrishnan, R.Chandramouli & N.Ranganathan, "Functional Reconfiguration for Fault-Tolerance: A New Approach", Department of Computer Science & Engineering, Center for Microelectronics Research, University of South Florida

[١١] R. D. (Shawn) Blanton, Seth Copen Goldstein, Herman Schmit, "Tunable Fault Tolerance via Test and Reconfiguration", Dept. of Electrical and Computer Engineering, Dept. of Computer Science, Carnegie Mellon University

[١٢] Brian Logan, "Classifying Agent Systems", School of Computer Science, University of Birmingham, UK

[١٣] Cesar Ortega and Andy Tyrrell, "Biologically Inspired Fault-Tolerant Architectures for Real-Time Control Applications", Department of Electronics, University of York, York, YO10 5DD, UK, Published in Control Engineering Practice, July 1999, pp. 673-678

[١٤] Daryl Bradley, Cesar Ortega-Sanchez, Andy Tyrrell, "Embryonics + Immunotronics: A Bio-Inspired Approach to Fault Tolerance", Department of Electronics, University of York, Heslington, York, YO10 5DD, UK.

[١٥] L. E. Parker, "ALLIANCE: architecture for fault tolerant, cooperative control of heterogeneous mobile robots," in Proc. 1994 IEEE/RSJ/GI Int. Conf. Intelligent. Robot. Systems. (IROS '94), Munich, Germany, Sept. 1994, pp. 776—783

[١٦] Deen, S.M., "A Fault-tolerant Cooperative Distributed System", Proceedings Ninth International Workshop on Database and Expert Systems Applications, 1998, pp. 508-513

[١٧] Vogler, H., Kunkelmann, T., Moschgath, M.-L., "An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions", Proceedings. 1997 International Conference on Parallel and Distributed Systems, 1997, pp. 268-274

[١٨] Vogler, H., Kunkelmann, T., Moschgath, M.-L., "Distributed transaction Processing as a Reliability Concept for Mobile Agents", Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1997, pp. 59-64

[۱۹] Strasser, M., Rothermel, K., "Reliability concepts for mobile agents", Int. J. Coop. Inf. Syst. (Singapore), vol.7, no.4, Dec. 1998, pp. 355-82

[۲۰] Stoller, S.D., Schneider, F.B., "Automated stream-based analysis of fault-tolerance", Formal Techniques in Real-Time and Fault-Tolerant Systems. 5th International Symposium, FTRTFT'98. Proceedings, 1998, pp. 113-122

[۲۱] Assis Silva, F.M., Popescu-Zeletin, R., "An approach for providing mobile agent fault tolerance", Mobile Agents. Second International Workshop, MA'98. Proceedings, 1998, pp. 14-25

[۲۲] Witting T., ed., "ARCHON an Architecture for multi-agent systems", Ellis Horwood, 1992

[۲۳] Kasahara, H., Takadama, K., Nakasuka, S., Shimohara, K., “Fault tolerance in a multiple robots organization based on an organizational learning model”, SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics, vol.3,5, vol. 4945, 1998, pp. 2261-2266

[۲۴] فواد قادري، تحمل‌پذيري‌خرابي در مأموريت بلند كردن اجسام توسط رباتهاي همكار، پايان‌نامه كارشناسي ارشد، دانشگاه تهران، مهر ۱۳۸۰

[۲۵] Brooks, R. A., "New Approaches to Robotics" *Science*, Vol. 253, September 1991, pp. 1227--1232.

[۲۶] and Operating Krithi Ramamritham, John A. Stankovic. "Scheduling Algorithms Systems Support for Real-time Systems", Proceedings of the IEEE, 82(1):55-66, 1994.

[۲۷] A.L. Liestman, R.H. Campbell, "A Fault Tolerant Scheduling Problem, IEEE Transactions on Software Enginnering", SE-12(11):1089-95, Nov.1986

[۲۸] C.M.Krishna, K.G.Shin, "On Scheduling Tasks with a Quick Recovery from Failure", IEEE Transactions on Computers, C-35(5):448-55, May 1986

[۲۹] Manimaram, G. and Ram Murthy, C. S., "A Fault-tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-time Systems and Its Analysis", IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 1, 1998, pp. 1137-1152.

[۳۰] F. Liberato, R. Melhem and D. Mosse, "Tolerance to Multiple Transient Faults for Aperiodic Tasks in Hard Real-time Systems", IEEE Trans. on Computers, Vol 49, No 9, Sep 2000

[۳۱] D. Mosse, R. Melhem and S. Ghosh, "Analysis of a Fault-Tolerant Multiprocessor Scheduling Algorithm". Proc. of the 24th Fault-Tolerant Computing Symposium. Austin, TX. (1994).

[۳۲] Anup K. Bhattacharjee, K. Ravindranath, A. Pal, R. Mall, "DDSCHED: a distributed dynamic real-time scheduling algorithm", Progress in Computer Research Book, ACM Portal, Pages: 170 - 184, ISBN:1-69033-010-2, 2001

[۳۳] S. Swaminathan, G. Manimaran: "A Reliability-Aware Value-Based Scheduler for Dynamic Multiprocessor Real-Time Systems", 16th International Parallel and Distributed Processing Symposium (IPDPS 2002),Apr. 2002, FL, USA, IEEE Computer Society 2002, ISBN 0-7695-1573-8

[۳۴] دكتر صديقي مشكناني، دكتر حسين پدرام،" اصول طراحي و ويژگي‌هاي داخلي سيستم‌هاي عامل"، ويرايش سوم ۱۹۹۸، نويسنده: ويليام استالينگ، نشر شيخ بهايي.

[۳۵] K. Tindell, A. Burns, and A.J. Wellings. "Allocating real-time tasks (an NP-hard problem made easy)". *Real-Time Systems*, 4(2):145-165, Jun 1992.

[۳۶] Noga Alon, Yossi Azar, Gerhard J. Woeginger, Tal Yadid, "Approximation Schemes for Scheduling", Eighth Annual ACM-SIAM Symposium on Discrete Algorithms(SODA),New Orleans, Louisiana, Jan 1997, pp493-500.

[۳۷] Ola Redell, "Global Scheduling in Distributed Real-time Computer Systems, An Automatic Control Perspective", Technical Report, Department of Machine Design, Royal Institute of Technology, ISSN 1400-1179

[۳۸] N. Audsley, A. Burns, "Real-Time System Scheduling", Department of Computer Science, University of York, UK.

[۳۹] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, " A New Task Redistribution Method for Fault Clearing in Multi-agent Systems", Proc. of IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, October 6-9, 2002, Hammamet, Tunisia

[۴۰] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, "Evaluating Task Redistribution Methods for Fault Clearing in Multi-Agent Systems", ERSA'02, 2002 pp219-225, Las Vegas, USA, May

[۴۱] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, "Agent's Role Reconfiguration Based on Decision-Making for Distributed Fault Recovery in Multi Agent Systems", AIM'02, Workshop of EURASIA-ICT 2002, pp 289-293, Shiraz, Iran, Oct. 2002.

[۴۲] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, "A Decision-Making based Approach for Fault-Handling in Multi-Agent Systems", ICONIP'02, Vol.4, pp 1905-1910, Singapore, Nov. 2002.

[٤٣] مریم سادات میریان, مجید نیلی احمدآبادی, یک نگرش چندعامله گسترده برای افزایش تحمل پذیری خطا, هشتمین کنفرانس سالانه انجمن کامپیوتر ایران, مشهد, ۶ تا ۸ اسفند, صفحه ۴۱۳ تا ٤١٩

[٤٤] Maryam S. Mirian, Majid Nili Ahmadabadi, Babak Nadjar Araabi, "A Fault Tolerant Multi-Agent System with Non-Deterministic Decision-Making for Task Allocation", Accepted in ERSA'03, June 23-26, 2003, Las Vegas, Nevada, USA.

[٤٥] Maryam S. Mirian, Majid Nili Ahmadabadi, "Help-based Distributed Scheme for Fault Tolerant Deterministic Systems", Submitted to IROS2003, March2003.

[۴۶] Maryam S. Mirian, Majid Nili Ahmadabadi, "Help Provision, a Mechanism to Improve the Fault Tolerance of Distributed Systems", To be Submitted to International Journal of Autonomous Agents and Multi-Agents Systems, May 2003

[۴۷] -in Behavior Oriented Multi-Robot Learning- Task:ALLIANCE-Lynn E. Parker, "L Based Systems Advanced Robotics", Special Issue on Selected Papers from IROS '96, 11 (4) 1997: 305-322.

فهرست