ارائه روشی جدید برای شناسایی و از بین بردن پراسس‌های یتیم

آرش صباغی[1*]

[1]مربی گروه کامپیوتر، واحد سمنان، **دانشگاه آزاد اسلامی**، سمنان، ایران
a.sabbaghi@semnaniau.ac.ir *

چکیده

در سیستم‌های توزیع شده به دلیل انجام فراخوانی پروسیجرهای راه دور، امکان ایجاد پراسس‌های یتیم وجود دارد. در واقع پراسس‌های یتیم، محاسبات راه دوری هستند که فراخواننده آنها به هر دلیل متوقف شده باشد. وجود محاسبات یتیم در سیستم به صورت کلی مطلوب نیست چراکه منجر به هدر رفت منابع می‌شوند و همچنین این امکان وجود دارد که وضعیتهای ناسازگاری را در داده اشتراکی ببینند. در این مقاله روشی جدید به منظور شناسایی و از بین بردن پراسس‌های یتیم ارائه شده است. این روش دارای مزایای قابل توجهی نسبت به روش‌های مطرح موجود در این حوزه می‌باشد. از مهم ترین مزایای این روش می‌توان به این موارد اشاره نمود: از بین بردن تمامی فرزندان و نوادگان راه دور پراسس یتیم ،تعداد اندک پیام ،عدم کاهش کارایی،توازن بار و افزایش قابلیت تحمل خطا.

کلید واژگان

پراسس‌های یتیم، سیستم‌های توزیع شده، فراخوانی پروسیجر راه دور.

A New Approach to Detect and Eliminate Orphan Processes

Arash Sabbaghi [1]

1- Department of Computer Engineering, Semnan Branch, **Islamic Azad University**, Semnan, Iran.
a.sabbaghi@semnaniau.ac.ir

Abstract

In distributed systems, due to performing remote procedure calls, there is the possibility of generating orphan processes. Actually orphan processes are remote computations which their callers are crashed for any reason. The existence of orphan computations are not desirable because they waist resources and also there is the possibility that they see inconsistent situations in shared data. In this paper, a new approach in order to detect and eliminate orphan processes is presented. Proposed approach has significant advantages over existing methods in this area. The main advantages of this method are as follows: detection and elimination of all remote children and grandchildren of orphan process, few numbers of communication messages, no decrease in efficiency, load balancing and increase in fault tolerance.

Keywords

Orphan Processes, Distributed Systems, Remote Procedure Call

۱- مقدمه

یک پراسس می‌تواند تمام محاسبات مربوط به خود یا قسمتی از آن را به یک سرور جهت اجرا، ارجاع دهد. این پراسس، پراسس فراخواننده و محاسبات مربوط به آن در سرور، محاسبات یا پراسس‌های فراخوانده شده نامیده می‌شوند. اگر به هر دلیلی پراسس فراخواننده متوقف شود، اجرای محاسبات مربوط به آن در سرور ادامه خواهد یافت، درحالیکه این محاسبات بی فایده است و پراسسی منتظر پایان اجرا و نتایج آنها نیست. در این حالت به این پراسسِ فراخوانده شده، پراسس یا محاسبات یتیم می‌گویند.

محاسبات یتیم برای سیستم به دو دلیل عمده مطلوب نیستند: اولا، یتیم‌ها منابع سیستم را هدر می‌دهند، از سیکل‌های پراسسور استفاده می‌کنند، بافر پیغام‌ها را اشغال می‌کنند و می‌توانند بر روی داده‌ها، قفلهایی ایجاد کنند که موجب به وجود آمدن تاخیر در پراسس‌های بعدی شود. ثانیا، یتیم‌ها ممکن است وضعیت‌های ناسازگاری را در داده اشتراکی ببینند. این مسئله می‌تواند برنامه‌ها را جهت طراحی و ساخت دچار مشکل کند. از این رو باید به نحوی این پراسس‌های یتیم را اداره نمود و اثرات جانبی آنها را به حداقل رساند.

مطالعات بر روی محاسبات یتیم در حوزه‌های زیر انجام شده است: شناسایی پروسس‌های یتیم وکشتن آنها و یا از بین بردن اثرات آنها، جلوگیری از دیدن داده‌های ناسازگار توسط یتیم‌ها و جلوگیری از ایجاد پراسس‌های یتیم.

در بسترهای مختلفی امکان بروز پراسس‌های یتیم وجود دارد: تراکنش‌های عادی و تو در تو، فراخوانی پروسیجر راه دور RPC، عامل‌های سیار و فراخوانی تکرار شده. در ادامه به هر کدام مختصرا می‌پردازیم.

کارهای اولیه جهت جلوگیری از دیدن داده‌های ناسازگار، الگوریتم‌های شناسایی و حذف یتیم‌هایی را پوشش می‌دادند که از طریق خرابی[1] نودها حاصل شده بودند[1]. در این مطالعات مکانیزمهای زیرین تراکنش‌ها از جمله همروندی در نظر گرفته نشده بود. به همین دلیل ارائه یک مفهوم ساده جهت متوقف کردن محاسبات یتیم، بسیار دشوار بود.

[1] Crash

در کارهای بعدی یتیم‌ها در قالب تراکنش‌های تودرتو[1] مورد بررسی قرار گرفتند[2]. از طریق تراکنش‌های تو در تو می‌توان محاسبات را در سیستم‌های توزیع شده سازماندهی کرد. این نوع تراکنش‌ها علاوه بر اینکه مانند تراکنش‌های عادی یک ساختار ساده جهت پوشاندن اثرات همروندی و خرابی‌ها فراهم می‌آورند، اجازه همروندی در یک تراکنش تنها را نیز فراهم می‌آورند. آنها همچنین درجه بالاتری از تحمل پذیری در برابر خطا[2] را به وسیله مجزا سازی تراکنش از خرابی‌های پدرانش فراهم می‌آورند. در تراکنش‌های تودرتو محاسباتی که قرار است متوقف شوند به سادگی انصراف می‌یابند و از تاثیر آن بر روی وضعیت سیستم جلوگیری می‌شود. هدف این فعالیت‌ها تشخیص و حذف یتیم‌ها قبل از دیدن داده‌های نا سازگار است. الگوریتم‌های متنوعی برای جلوگیری از محاسبات یتیم در سیستم‌های مبتنی بر تراکنش ارائه شده است. این الگوریتم‌ها از طریق استخراج خصوصیات جریان داده در سیستم مدیریت تراکنش[3]، تضمین می‌کنند که یتیم‌ها فقط وضعیتهایی را در داده اشتراکی می‌بینند که اگر یتیم نبودند، می‌دیدند. هنگامی‌که این الگوریتم‌ها با الگوریتم‌های صحیح کنترل همروندی ترکیب شوند و مورد استفاده قرار گیرند، تضمین می‌کنند که همه محاسبات (چه یتیم باشند و چه عادی) وضعیت‌های سازگار داده اشتراکی را می‌بینند.

یکی دیگر از بسترهای بروز پراسس‌های یتیم RPC است که در سیستم‌های توزیع شده مورد استفاده قرار می‌گیرد. به عنوان مثال یک نود که درخواست راه دور ایجاد می‌کند ممکن است به علت مشکلات شبکه، مانند: گم شدن پیام‌ها و خرابی نودها یا مشکلات دیگر نتواند ارتباط خود را با نودهای دیگر حفظ کند. در این حالت ممکن است پراسس‌های یتیم در نود فراخوانده شده ایجاد گردد[3].

حوزه دیگر بروز یتیم‌ها، عامل‌های سیار[4] هستند[4]. روش‌های شناسایی و حذف یتیم‌ها در سیستم‌های توزیع شده، از روابط بین پدر-پسر که یک سیستم توزیع شده تعریف می‌کند استفاده می‌نمایند اما این روشها در سیستم‌های عامل سیار قابل اعمال نیستند، علت این امر نیز این است که یک رابطه مشابه طبیعی بین عامل‌ها وجود ندارد. بنابراین پروتکل‌های جدیدی برای آنها باید گسترش یابند.

حیطه چهارم رخداد محاسبات یتیم، در فراخوانی تکرار شده[5] می‌باشد[5]. در سیستم‌های امروزی ، برنامه‌های کاربردی از اجزای[6] مختلفی تشکیل یافته اند که این اجزاء میتوانند در یک جا مجتمع شده باشند یا بر روی ماشین‌های متفاوت قرار گرفته باشند . این اجزاء به منظور پاسخ گویی به درخواست کلاینت، با یکدیگر همکاری می‌نمایند. برای تضمین عملکرد برنامه در صورت بروز خرابی، عموما از تکرار اجزاء بهره می‌گیرند. تکرار اجزاء به این مشکل منتهی می‌گردد که یک سرور تکرار شده، سرور تکرار شده دیگری را فراخوانی نماید. به این مساله ، فراخوانی تکرار شده می‌گویند. این فراخوانی‌های تکرار شده موجب بروز درخواست‌های یتیم می‌شوند.

در این مقاله روشی برای شناسایی و از بین بردن محاسبات یتیم در حوزه RPC ارائه گردیده است. مزیت اصلی این روش نسبت به روش‌های مطرح موجود در این حوزه این است که این روش، علاوه بر حذف پراسس یتیم، امکان حذف تمامی فرزندان ، نوادگان و... آن پراسس یتیم را نیز فراهم می‌آورد. این روش از تعداد اندکی پیام‌های ارسالی استفاده می‌کند و سبب کاهش کارایی نمی‌گردد. از طرف دیگر توازن بار و افزایش قابلیت تحمل خطا را به همراه دارد.

در ادامه، در بخش ۲ به مطالعاتی که در گذشته برای شناسایی و حذف یتیم‌ها در حوزه RPC انجام شده است، می‌پردازیم. در بخش۳، روش پیشنهادی و مزایای آن تشریح می‌گردد. در بخش۴، مقایسه ای بین روش پیشنهادی و روش‌های موجود انجام شده است و بخش۵، به نتیجه گیری می‌پردازد.

۲- کارهای گذشته

مطالعات مختلفی در حوزه تشخیص و شناسایی پراسس‌های یتیم صورت گرفته است[8,9,10,11]. اولین فردی که به صورت دقیق، به شناسایی یتیم‌ها پرداخت، شخصی به نام Nelson بود که در سال ۱۹۸۱ چهار روش جهت از بین بردن یتیم‌ها ارائه کرد [6]:

روش براندازی / نابودی[7] . در این روش، stub کلاینت پیش از ارسال یک پیام RPC، یک رکورد log درباره آنچه که می‌خواهد انجام دهد را ثبت می‌کند. logها بر روی دیسک یا بر روی رسانه‌های دیگری که در مقابل خرابی مصون هستند، نگه داری میشوند. کلاینت بعدا می‌تواند مجددا راه اندازی شود و از روی log خود (گزارش ثبت وقایع پیش از انجام آنها[8]) ، درخواست‌هایی که جواب ندارند را شناسایی کرده و به سرور مربوطه اعلام کند که آنها را بکشد. در واقع این روش، کشتن یتیم‌ها را به پدرشان می‌سپارد. این روش خالی از ایراد نیست: اولا، حجم زیادی از اطلاعات (یک رکورد برای هرRPC) باید بر روی دیسک نوشته شود و سبب می‌گردد که سربار نوشتن بر روی دیسک زیاد گردد. ثانیا این روش در صورتیکه یتیم‌ها خودشان RPC انجام دهند، کارایی نخواهد داشت. چرا که شناسایی فرزندان یتیم‌ها در این روش، سخت یا غیر ممکن است و در نهایت ممکن است شبکه به سبب خرابی دروازه‌ها به چندین بخش تقسیم گردد، در این صورت کشتن یتیم‌ها علی رغم شناسایی آنها غیرممکن می‌گردد.

روش تناسخ[9] . در این روش بیشتر مشکلات روش براندازی بدون نیاز به نوشتن رکوردهایی بر روی دیسک، حل می‌گردد. در این روش مفهومی به نام دوره زندگی[10] ، وجود دارد. بدین صورت که زمان به بخش‌هایی به نام دوره تقسیم می‌گردد که این دوره‌ها دارای شماره می‌باشند. زمانیکه یک کلاینت مجددا راه اندازی می‌گردد، یک پیام را به تمامی ماشین‌ها همه پخشی می‌کند که بیانگر آغاز یک دوره جدید می‌باشد. زمانیکه چنین پیامی دریافت می‌شود، تمام پراسس‌های راه دور کشته می‌شوند. بنابراین در صورتیکه شبکه به چندین بخش تقسیم گردد، تعدادی از یتیم‌ها می‌توانند از نابود شدن مصون بمانند. در این روش با آغاز هر دوره، تمام فرزندان حتی اگر پدر داشته باشند کشته می‌شوند.

روش تناسخ آرام[11]. این روش تعمیم یافته روش تناسخ است. زمانیکه پیام آغاز یک دوره جدید دریافت می‌گردد، هر ماشین بررسی می‌کند که آیا پروسس‌های راه دور داشته است یا خیر، در صورت وجود، سلامت مالک آنها

[1] Nested Transaction
[2] Fault Tolerance
[3] Transaction Manager
[4] Mobile Agents
[5] Replicated Invocation
[6] Component
[7] Extermination
[8] Write-ahead-log
[9] Reincarnation
[10] Epoch
[11] Gentle Reincarnation

را بررسی می‌کند. تنها در صورتیکه مالک آنها خراب شده باشد ، پراسس‌ها کشته می‌شوند.

روش انقضا[1]. در این روش هر درخواست RPC یک زمان استاندارد به اندازه T دارد که اجازه دارد در این زمان کار خود را انجام دهد. اگر کار آن در این زمان پایان نیافت، باید صراحتا یک کوانتوم دیگر به اندازه T درخواست نماید. با این کار پراسس به سرور اعلام می‌کند که سالم است و انجام کار خود را به اندازه زمان T تمدید می‌کند. به عبارت دیگر اگر کلاینت پس از یک خرابی، به اندازه زمان T قبل از راه اندازی مجدد صبر کند، تمامی یتیم‌ها مطمئنا کشته می‌شوند. چالش این روش انتخاب مقدار منطقی و صحیح برای T با در نظر گرفتن RPC و نیازمندیهای متفاوت آن است.

در [3]، سه روش جهت شناسایی و حذف محاسبات یتیم ارائه شده است.

روش اول: هر فراخوانی شامل یک زمان نهایی[2] است که حداکثر زمان در دسترس جهت اجرا را تعیین می‌کند. اگر این زمان فرا برسد سرور اجرا را Abort کرده و فراخوانی به صورت غیر عادی خاتمه می‌یابد. به این ترتیب اگر هیچ خرابی در سیستم رخ ندهد، می‌توان مطمئن بود که تمام امکانات لازم جهت مقابله با یتیم‌ها وجود دارد.

روش دوم: هر نود یک شمارنده که Crash Count نامیده میشود، دارد که بلافاصله بعد از اینکه نود از حالت خرابی خارج شد، یک واحد اضافه می‌شود. همچنین هر نود دارای یک جدول از مقادیر Crash Count کلاینتهایی که با او تماس برقرار کرده اند، می‌باشد. در درخواست تماس، مقدار Crash Count تماس گیرنده یا کلاینت گنجانده می‌شود. اگر این مقدار از مقدار موجود در جدول سرور بزرگتر باشد، متوجه می‌شود که در کلاینت خرابی رخ داده و ممکن است از درخواست‌های قبلی این کلاینت، یتیم‌هایی در سیستم باقی مانده باشد. به این ترتیب ابتدا این یتیم‌ها شناسایی شده و قبل از شروع اجرای فراخوانی Abort می‌شوند.

روش سوم: این روش تعمیم یافته روش دوم است. در این روش، هر نود یک پراسس خاتمه دهنده[3] دارد که به صورت تناوبی مقادیر Crash Count نودهای دیگر را بررسی کرده و هر نوع یتیم را در صورت تشخیص، abort می‌کند.

روش‌های اول و دوم می‌توانند با سربار کمی، که به کارایی صدمه چندانی نمی‌زند، اجرا گردند. روش سوم تا حدی سربار به سیستم تحمیل می‌کند که می‌توان آنرا با افزایش بازه زمانی ارسال پیام‌ها کاهش داد. ولیکن در هر سه روش، تمامی‌پراسس‌های یتیم شناسایی و حذف نمی‌گردند، چرا که امکان شناسایی و حذف فرزندان یتیم‌هایی که در سرورهای دیگر هستند (فرزندان و نوادگان یتیمی که خود فراخوانی راه دور انجام داده اند)، وجود ندارند.

در [7]، دو روش دیگر جهت شناسایی و حذف محاسبات یتیم ارائه شده اند.

روش اول، گروه سرور اختصاص یافته (DSG[4]) ، نام دارد. در این روش هر کلاینت به یک سرور مشخص اختصاص یافته است، بنابراین درخواست کلاینت‌ها تنها به سرور مشخص شده می‌تواند ارسال گردد. از این رو، پس از راه اندازی مجدد کلاینت از خرابی، کلاینت تنها به سرورهای اختصاص یافته پیام آغاز دوره جدید را ارسال می‌نماید. در این روش نیز به فراخوانی‌های پروسیجر راه دور فرزندان و نوادگان یک پراسس یتیم توجه نشده است و آنها حذف نمی‌گردند.

روش دوم، انقیاد دهنده[5] می‌باشد. یکی از روش‌ها برای مکان یابی سرورها، استفاده از انقیادپویا است. در این روش، سرورها خود را به انقیاد دهنده معرفی می‌کنند . زمانیکه یک کلاینت می‌خواهد یک RPC را اجرا کند، stub کلاینت، یک پیام به انقیاد دهنده ارسال می‌کند و انقیاد دهنده یک سرور را انتخاب و آدرس آن را برای کلاینت ارسال می‌کند. ایده ارائه شده در [7] برای شناسایی و حذف یتیم‌ها بدین صورت است که: کلاینت پس از راه اندازی مجدد، پیام آغاز دوره جدید را به انقیاد دهنده ارسال می‌کند. از آنجائیکه سرورها، در زمان معرفی خود، کلاینت‌هایی که پشتیبانی می‌کنند را ثبت می‌کنند، انقیاد دهنده، پیام آغاز دوره جدید را تنها به سرورهای مرتبط ارسال می‌کند. مزیت این روش در این است که به جای ارسال پیام آغاز دوره جدید به کل شبکه، این پیام تنها به انقیاد دهنده ارسال می‌شود و انقیاد دهنده، پیام را تنها به سرورهای مشخص شده ارسال می‌کند. یک ایراد این روش در این است که متمرکز بوده و می‌تواند به گلوگاه تبدیل شود. این مشکل با افزونگی قابل حل است. در این روش نیز، مشکل از بین نرفتن فرزندان و نوادگان یتیم‌ها به قوت خود باقی است.

روش ارائه شده در این مقاله، تعمیم یافته روش انقیاد دهنده است که با ارائه راهکارهایی نوین تمامی مشکلات (گلوگاه و از بین نرفتن فرزندان و نوادگان یتیم‌ها) این روش را برطرف نموده و از طرف دیگر قابلیت تحمل پذیری خطا را به صورت قابل توجهی افزایش داده است.

۳- روش پیشنهادی

در این روش که ما آن را انقیاد دهنده توسعه یافته نامیده ایم، در ابتدا هر سرور خودش را به یک انقیاد دهنده، معرفی می‌کند. زمانیکه که یک کلاینت مانند C_1 درخواست خود را به انقیاد دهنده می‌دهد، انقیاد دهنده بر اساس مشخصه‌های سرورهایی که خود را معرفی کرده اند و میزان بهره وری آنها، آدرس سرور با پائین ترین مقدار بهره وری را به کلاینت، ارسال می‌کند. سپس انقیاد دهنده، اطلاعات این انقیاد را در جدول انقیاد خود ثبت می‌کند(نام کلاینت درخواست کننده، نام پراسس پدر، نام سرور اجرا کننده و نام پراسس). در صورتیکه یک پراسس خود ریشه باشد و پدر نداشته باشد، مقدارRP[6] در ستون پراسس پدر برای آن درج می‌گردد. جدول(۱)، اطلاعات انقیادهای صورت پذیرفته فرضی را نشان می‌دهد.

در این روش، هر گاه یک پراسس بخواهد فرزند ایجاد کند، سروری که محاسبات پراسس را انجام می‌دهد، مسئول ایجاد فرزند برای آن می‌باشد. سرورها برای ایجاد فرزند برای پراسسی که از کلاینت رسیده است، باید به انقیاد دهنده این درخواست را ارسال کنند. هر سرور یک متغیر به نام ChildCount برای هر پراسس دارد، که تعداد فرزندان آن پراسس را در آن ذخیره می‌کند. بر اساس مقدار این متغیر، سرور می‌تواند نام پراسس فرزند را تعیین کند(برای مثال، نام اولین فرزند پراسس P_1 ، $P_{1.1}$ می‌گردد). در ادامه، انقیاد دهنده توازن بار انجام می‌دهد و می‌تواند همان سرور یا سرور دیگری را برای اجرا برگزیند. اطلاعات این انقیاد نیز در جدول انقیاد ذخیره می‌گردد.

[1] Expiration
[2] Time Out
[3] Terminator
[4] DSG : Dedicated Server Group
[5] Binder
[6] RP : Root Process

جدول۱ نمونه ای از جدول انقیادهای صورت پذیرفته در یک انقیاد دهنده

نام پراسس	اجرا کننده	پراسس پدر	درخواست کننده
P_1	S_1	RP	C_1
$P_{1.1}$	S_2	P_1	S_1
$P_{1.2}$	S_3	P_1	S_1
$P_{1.2.1}$	S_2	$P_{1.2}$	S_3
P_2	S_3	RP	C_2
P_3	S_2	RP	C_1

پس از پایان یافتن اجرای یک پراسس، کلاینت پیامی را به انقیاد دهنده ارسال می‌کند و انقیاد دهنده تمامی رکوردهایی که مربوط به این پراسس و فرزندان آن می‌باشد را از جدول حذف می‌کند، بدین ترتیب جدول تنها پراسس‌های در حال اجرا را در خود خواهد داشت که سبب می‌گردد حافظه کمتری مصرف گردد و از طرفی سرعت جستجو افزایش یابد.

در صورتیکه کلاینت C_1، خراب شود، بعد از راه اندازی مجدد تنها نیاز به یک پیام دوره جدید دارد که آن را به انقیاد دهنده ارسال می‌کند و همانند روش تناسخ نیاز به همه پخشی در شبکه وجود ندارد. انقیاد دهنده سطرهای جدول را بررسی می‌کند و با توجه به ستون "درخواست دهنده"، درخواست‌های مربوط به این کلاینت را می‌یابد و به سرورهایی که درخواست به آنها ارجاع داده شده، اطلاع می‌دهد که پراسس‌های مربوطه را بکشند. در مرحله بعد بررسی می‌کند که آیا این درخواست در ستون دوم به عنوان پراسس پدر حضور دارد یا خیر، یعنی آیا این پراسس فرزندی دارد یا خیر. در صورت وجود فرزند، به سرور این پراسس‌ها اطلاع داده می‌شود و این الگوریتم به صورت بازگشتی تکرار می‌گردد تا اطمینان حاصل شود که فرزندان و نوادگان پراسس کلاینت توسط سرورهای مربوطه از بین رفته اند.

در این روش اگر یک سرور نیز خراب شود، بعد از بازیابی و ارسال پیام به انقیاد دهنده‌های خود، تمامی فرزندان و نوادگان راه دور پراسس‌های آن سرور را که قبل از خرابی در حال اجرا بودند نیز توسط روند مشابهی که ذکر شد، از بین می‌روند. همانطور که مشاهده می‌کنید این روش جدید با نوآوری خود توانسته با کمترین تعداد پیام‌ها، تمامی یتیم‌ها و فرزندان و نوادگان آنها را در هر سطحی از بین ببرد و در شرایط و خرابی‌های گوناگون قابل اعمال باشد و سبب کاهش بهره وری نیز نگردد. شبه کد پروسیجرهای اساسی این روش در شکل۱ نشان داده شده است. همچنین شکل۲، روند انقیادهای صورت پذیرفته برای پراسس P_1 و فرزندان و نوادگان آن را که در مثال جدول۱ آمده است را نشان می‌دهد.

برای جلوگیری از گلگوگاه شدن انقیاد دهنده در روش انقیاد دهنده توسعه یافته، می‌توان تعداد انقیاد دهنده‌ها را افزایش داد، که این افزایش خود به دو طریق امکان پذیر است:

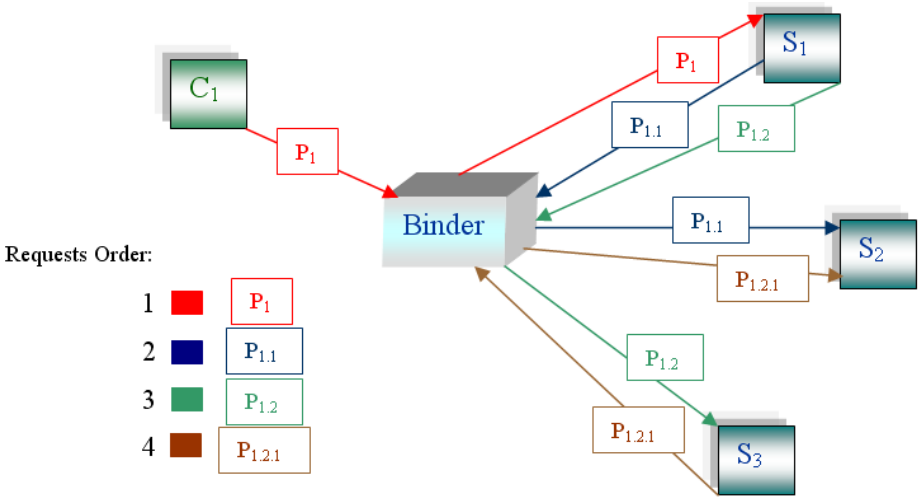
روش انقیاد دهنده توسعه یافته توزیع شده. در این روش به جای یک انقیاد دهنده ، چندین انقیاد دهنده خواهیم داشت که هر کدام به صورت مجزا و کاملا توزیع شده عمل می‌کنند. هر سرور خود را به یک انقیاد دهنده معرفی می‌کند.

```
Procedure Crash(C as Client)   //Calls when binder receive a crash message from a client
 {
        -For each   C  value  in  RequesterName.BindingTable   Do
                -Call  KillOrphanProcess( ProcessName.BindingTable , ServerName.BindingTable)
         EndFor
 }

Procedure KillOrphanProcess (P as Process, S as Server)   //Kill process P in server S & all the
children of process P
 {
        -Send a message to server S to kill process P
        -Delete binding table row contains values P as process name & S as server name
        -For each   P  value  in  FatherName.BindingTable   Do
                -Call  KillOrphanProcess( ProcessName.BindingTable , ServerName.BindingTable)
         EndFor  //This loop kill all the children's of process P an themselves children recursively
 }

Procedure NewChild (P as Process)  //Every process want to make a child call this procedure in its
server
 {
        -ChildCount(P)+=1
        -ChildProcessName = P.Name +  "."  + ChildCount(P)
        -Requests binder to bind process ChildProcessName with father P to a server
 }
```

شکل۱ شبه کد پروسیجرهای اساسی روش انقیاد دهنده توسعه یافته



شکل۲ روند انقیاد یک پراسس پدر و فرزندان آن به سرورها توسط روش انقیاد دهنده توسعه یافته

اگر پس از طی زمان T هیچ پراسسی جهت اجرا به او سپرده نشد، سالم بودن انقیاد دهنده را با یک پیام بررسی می‌کند، در صورت خراب بودن انقیاد دهنده، خود را به انقیاد دهنده دیگری معرفی می‌کند. در این روش هر انقیاد دهنده توازن بار را به صورت محلی انجام می‌دهد، بدین معنا که بین سرورهایی که خود را به آن انقیاد دهنده معرفی کرده اند، توازن بار انجام می‌دهد. این روش، به هیچ یک از انقیاد دهنده‌ها سربار اضافی تحمیل نخواهد کرد و سبب افزایش تحمل پذیری خطا می‌گردد.

روش انقیاد دهنده توسعه یافته تکرار شده. این روش همانند روش انقیاد دهنده توسعه یافته توزیع شده است، با این تفاوت که انقیاد دهنده‌ها به صورت مجزا عمل نمی‌کنند و از انقیادهای انجام یافته توسط یکدیگر باخبر هستند. مقادیر جدوال انقیاد آنها با به روز رسانی، مشابه یکدیگر می‌گردد. از مزایای این روش این است که خراب شدن یک انقیاد دهنده هیچ تاثیری در از بین بردن فرزندان و نوادگان پراسس‌های یتیمی که او انقیاد داده است، ندارد که این امر سبب می‌شود تحمل پذیری خطا در این روش به بالاترین حالت خود برسد. این روش مقداری سربار به همراه دارد که به سبب به روز رسانی و یکسان سازی جداول انقیاد، انقیاد دهنده‌ها می‌باشد. از دیگر مزایای این روش این است که توازن بار در کل شبکه صورت می‌پذیرد.

۴- مقایسه روش پیشنهادی با روش‌های مطرح موجود

در این بخش، مقایسه ای بین روش‌های پیشنهادی در مقاله (روش انقیاد دهنده توسعه یافته توزیع شده ، روش انقیاد دهنده تکرار شده) با روش‌های موجود مطرح در شناسایی و از بین بردن یتیم‌ها در حوزه RPC، انجام شده است. جدول۲ این مقایسه را نشان میدهد. در این جدول،

N : تعداد کل سرورها

n : تعداد سرورها در یک گروه

b: تعداد انقیاد دهنده‌ها

می‌باشد. تعداد پیام‌ها در جدول(۲) ، تعداد پیام‌هایی است که برای از بین بردن یک پراسس یتیم به سبب خرابی کلاینت مبادله میشود، است. تنها برای روش تناسخ، تمامی پیام‌های لازم برای از بین بردن تمامی یتیم‌ها درج شده است. همان طور که در جدول ۲ مشاهده می‌کنید، از میان روش‌های موجود گذشته، تنها روش تناسخ است که قادر به از بین بردن تمامی فرزندان و نوادگان پراسس‌های یتیم می‌باشد که از یک سو به سبب همه پخشی و سربار بالای آن و از طرف دیگر به دلیل از بین رفتن تمامی پراسس‌های در حال اجرا و در روش تناسخ آرام به دلیل سربار پیام‌های ارسالی برای پی بردن به سلامت مالک هر پراسس، مناسب نمی‌باشد. بعلاوه روش تناسخ تنها قادر به از بین بردن فرزندان و نوادگان پراسس‌های یتیمی است که از خرابی کلاینت ناشی شده اند، در صورتیکه روش ارائه شده در این مقاله، جهت از بین بردن فرزندان و نوادگان پراسس‌های یتیم ناشی از خرابی کلاینت و نیز ناشی از خرابی سرور، کارایی خواهد داشت. علاوه بر مزیت بارز ذکر شده فوق، با نگاهی به جدول ۲ می‌توان سایر مزایای قابل توجه رویکرد پیشنهادی را نسبت به روش‌های مطرح موجود دریافت.

۵- نتیجه گیری

روش‌های مختلفی برای شناسایی و از بین بردن پراسس‌های یتیم ارائه شده اند که هر یک سعی در بهبود کارایی،کاهش استفاده از حافظه، افزایش قابلیت تحمل پذیری خطا و کاهش سربار پیام‌ها نموده اند. در این مقاله رویکرد جدیدی ارائه شده است که دارای مزایای قابل توجهی نسبت به روش‌های مطرح موجود در این زمینه می‌باشد. در این مقاله مقایسه کامل و دقیقی بین روش ارائه شده و روش‌های پیشین ارائه شده است. ازمهم ترین مزایای روش پیشنهادی می‌توان به تعداد بسیار کم پیام‌های ارتباطی، از بین بردن تمامی فرزندان و نوادگان راه دور پراسس یتیم، توازن بار و قابلیت تحمل پذیری خطای بالا اشاره کرد.

جدول۲ مقایسه ای بین روش‌های ارائه شده و روش‌های مطرح موجود

نام روش	تعداد پیام‌ها	مزایا	معایب
انقضاء	۱	۱- عدم نیاز به همه پخشی پیام‌ها	۱- عملیات RPC به کندی صورت می‌پذیرد ۲- مصرف حافظه دارد ۳- نیاز به الگوریتم جمع آوری زباله دارد ۴-Domino Effect دارد ۵- باقی ماندن فرزندان و نوادگان راه دور پراسس یتیم
تناسخ	حداکثر N	۱- از بین بردن فرزندان و نوادگان راه دور پراسس یتیم ۲- عملیات RPC به سرعت انجام می‌گیرد	۱- سربار زیاد به دلیل همه پخشی پیغام ۲- مشکل Exponential Roll Back وجود دارد.
انقیاد دهنده	۲	۱- عدم نیاز به log ۲- عدم نیاز به همه پخشی پیام‌ها ۳- انجام توازن بار	۱- وجود گلوگاه ۲- باقی ماندن فرزندان و نوادگان راه دور پراسس یتیم
گروه سرور اختصاص یافته	حداکثر ۲ * n	۱- عدم نیاز به log ۲- عدم نیاز به همه پخشی پیام‌ها ۳- توازن بار در کل شبکه ۴- عدم نیاز به الگوریتم جمع آوری زباله ۵- عدم وجود Domino Effect ۶- عدم وجود مشکل Exponential Roll Back	۱- باقی ماندن فرزندان و نوادگان راه دور پراسس یتیم
انقیاد دهنده توسعه یافته توزیع شده	۲	۱- از بین بردن فرزندان و نوادگان راه دور پراسس یتیم ۲-عدم نیاز به همه پخشی پیام‌ها ۳- توازن بارمحلی ۴- عدم نیاز به الگوریتم جمع آوری زباله ۵- عدم وجود Domino Effect ۶- عدم وجود مشکل Exponential Roll Back ۷- وجود حافظه پویا و مصرف بهینه حافظه ۸- عدم وجود گلوگاه ۹- دارای قابلیت تحمل پذیری خطا بالا	۱- مصرف حافظه (که به دلیل پویا بودن حافظه مصرفی، ناچیز می‌باشد)
انقیاد دهنده توسعه یافته تکرار شده	حداکثر b	۱- از بین بردن فرزندان و نوادگان راه دور پراسس یتیم ۲-عدم نیاز به همه پخشی پیام‌ها ۳- توازن بار در کل شبکه ۴- عدم نیاز به الگوریتم جمع آوری زباله ۵- عدم وجود Domino Effect ۶- عدم وجود مشکل Exponential Roll Back ۷- وجود حافظه پویا و مصرف بهینه حافظه ۸- عدم وجود گلوگاه ۹- دارای قابلیت تحمل پذیری خطا بسیار بالا	۱- مصرف حافظه (که به دلیل پویا بودن حافظه مصرفی، ناچیز می‌باشد) ۲- سربار همسان سازی جداول انقیاد دهنده‌ها

۶- مراجع

[1] A. Z. Spector, "*Performing remote operations efiiciently on a local computer network*," Commun. ACM, vol. 25, no. 4, pp. 246-260, Apr. 1982.

[2] MAURICE p. HERLIHY and MARTIN S. McKENDRY." *Timestamp-Based Orphan Elimination*" IEEE Transactions on Software Engineering. vol. IS. no. 7, July 1989.

[3] FABIO PANZIERI AND SANTOSH K. SHRIVASTAVA."*Rajdoot: A Remote Procedure Call Mechanism Supporting Orphan Detection and Killing*", IEEE Transactions on Software Engineering. vol. 14. No. I. January 1988.

[4] Joachim Baumann. "*A Protocol for Orphan Detection and Termination in Mobile Agent Systems*", IPVR (Institute for Parallel and Distributed High-Performance Systems) July 1997.

[5] Stefan Pleisch , Arnas Kup˘sys, Andr´e Schiper . **"***Preventing Orphan Requests in the Context of Replicated Invocation*". Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS'03). IEEE, 2003.

[6] Andrew S. Tanenbaum," *Distributed Operating System*",Prentice-Hall, 2003.

[7] M. Jahanshahi, K. Mostafavi, M. S. Kordafshari, M. Gholipour , A.T. Haghighat. **"***Two New Approaches for Orphan Detection*". Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05), IEEE, 2005.

[8] Dickman, Peter. "*Incremental, distributed orphan detection and actor garbage collection using graph partitioning and Euler cycles*." In *Distributed Algorithms*, pp. 141-158. Springer Berlin Heidelberg, 1996.

[9] Hagerstrom, Carl F., Thomas Dixon Hutchinson, Shridhar Bharthulwar, and Paul E. Tinius. "*Detecting and managing orphan*

files between primary and secondary data stores." U.S. Patent 7,685,177, issued March 23, 2010.

[10] Ezzat, Ahmed K. "*Orphan elimination in distributed object-oriented systems*." In *Distributed Computing Systems, 1990. Proceedings., Second IEEE Workshop on Future Trends of*, pp. 403-412. IEEE, 1990.

[11] O'donnell, Michael D., Danny B. Gross, and Gene R. Toomey. "Orphan computer process identification." U.S. Patent 6,480,877, issued November 12, 2002.