بازیافت شفاف در سیستمهای توزیع شده

1.مقدمه

ما راه حل های خوش بینانه ی شفاف را برای مشکلات سیستمهای توزیع شده همچون بازیافت،تکرار(تکثیر)،و موازی بودن وچاره های رقابت همزمان بررسی میکنیم.

با یک راه حل شفاف برای یک مسأله،ما آن را معنی میکنیم یک برنامه ای که به صورت اتوماتیک تبدیل شده است ورفتار آن برنامه مشابه رفتار یک برنامه ی تبدیل نشده است.علاوه بر این برنامه نویس و کاربرنهایی نیازی ندارند که از این تبدیل باخبر باشند. برای این قبیل مشکلات ،راه حل های شفاف نسبتأدرست هستنداگربر روی همزمان سازی تکیه کرده باشند.اما کارایی چنین متدهایی معمولاً ناچیز(ضعیف) است و همچنین پیاده سازی آن نیز گران است و شاخص نیستند.

روش (دیدگاه) ما استفاده از متدهای خوش بینانه ای است که فرض می کنیم همزمان سازی ضروری نیست و به این ناهمزمانی هنگامی رسیدگی میشود که برنامه به اجرا ادامه میدهد.ما روابط(وابستگیها ی) پردازه ی مدفون شده را پیدا می کنیم و رویدادهای غیرقطعی را ثبت می کنیم به طوری که بتوانیم برگردیم به حالت قبلی یک محاسبه که به یک فرض نادرست مربوط است.

2.*بازیافت خوش بینانه*

روش ما برای بازیافت مبنی بر بازیافت خوش بینانه است اضافه شده با بهینه سازی هایی برای کاهش اندازه ی ثبت و توسعه هایی که فایل سیستم و مؤلفه های خارجی(بیرونی)دیگر را داخل فرایند بازیافت ترکیب می کنند.

در بازیافت خوش بینانه فرض می کنیم پردازه ها خراب نیستند بویژه برای هر رویداد غیرقطعی(معمولاً یک پیام)فرض می کنیم معیوب نخواهد شد قبل از اینکه پیام آن به صورت ناهمزمان ثبت شود.هرکدام از این فرضها با یک شماره مشخص شده است و هر پردازه فرضی را که بیشترین شماره را از پردازه های دیگر دارد ثبت می کند بر اساس اینکه آن به کدام مربوط است.

در یک رویداد معیوب،پردازه ی معیوب دوباره شروع می شود وسپس همگام می شود با مجاورش با به عقب غلتیدن برای رسیدن به یک حالت سازگار (پایداری) دو طرفه (متقابل). بنابراین بازیافت در یک مکانیزم محافظه کار بسیار گران تر است. اما اجرای بدون خرابی اصولاً پیشرفت کرده است (بهبود یافته است) زیرا checkpointing وبسیاری از محاسبات می تواند به صورت ناهمزمان و با اجرای نرمال برنامه انجام شود.

3.*تراکنشها (نقل و انتقال ها) نارسا هستند*

این موقعیت ماست که تلرانس خطای شفاف مورد نیاز است زیرا سیستمهای مبتنی بر تراکنش برای موارد کاربری توزیع شده ی بزرگ نارسا هستند. برای این امر دو دلیل اصلی وجود دارد:

اولاً تراکنشها فقط داده را بازمی گردانند نه حالت فرایند را. این در محیط محاسبات متمرکز قابل قبول بود در تراکنشهایی که توسعه یافته بودند اما در یک سیستم توزیع شده ی بزرگ حالت مؤلفه های معیوب باید ترمیم شود (به حالت اول بازگردانده شود) تا تلرانس خطای آنها درست باشد . برای نمونه ،در یک برنامه ی کاربردی توزیع شده شامل مجموعه ای از نمایشگر ، محاسبه گر و کارسازدادگان(database servers) اگر یک گره پایگاه داده درهم شکند و دوباره شروع شود باید به هر طریقی دوباره ارتباطات آن را با مؤلفه های دیگر سیستم برپا کند (تجدید کند) و کاری را که نمایش داده شده است (اجرا شده است) قبول داشته باشد و فقط پس از آن برنامه ی کاربردی توزیع شده می تواند به اجرا ادامه دهد.

در یک مورد کاربری توزیع شده ، تراکنشها هیچ پشتیبانی از نوسازی یک حالت معیوب نمی کنند. در نتیجه،حتی اگر این همزمان سازی و توافق ها (قراردادها) به درستی برنامه ریزی شده باشد ، برنامه ی کاربردی خراب خواهد شد گرچه پایگاه داده خراب نشده باشد.

به همین دلیل، به درستی نوشتن تلرانس خطای برنامه ی کاربردی توزیع شده نیاز خواهد داشت به برنامه نویسی اضافه شدنی گسترده برای بازآفرینی حالات همزمان سازی داخل پردازه ای.

این کد پیچیده خواهد بود و احتمالش کم است که امتحانش (آزمودنش) خوب باشد چون آن در مسیر برنامه ی کاربردی اصلی نیست و تعداد روش های معیوب زیاد خواهد بود. در نتیجه کدی که کدی که برای تهیه ی تلرانس خطا فرض شده است کوچکترین مؤلفه ی معتبر بی عیب (یکپارچه) سیستم خواهد بود . از طرف دیگر این کد اغلب اوقات به سادگی حذف خواهد شد هنگامی که پس از اینکه یک قسمت زیادی از نرم افزار نوشته شده و یک برنامه ی کاربردی کامل (بی عیب) زمانی که با یک خطای پیش بینی نشده مواجه شود و بی نتیجه بماند.

مشکل دوم تراکنش ها این است که حتی زمانی هم که فقط بازیافت داده مورد نیاز است برنامه نویس باید به طور واضح برای خرابی برنامه ریزی کند با اطمینان از اینکه تراکنش ها بقدر کافی کوتاه هستند و جابجایی به وسیله ی آگاهی دادان به کاربر ، تجدید نظر کردن (دوباره آزمایش کردن) در تراکنش ها و... بی نتیجه می ماند.

راه حل های شفاف ومحدودیت های آنها

بازیافت خوش بینانه این مشکلات را حل می کند.در نتیجه،برنامه ها می توانند خیلی ساده تر شوند :

نیازی به ساختار برنامه های کاربردی همچون مجموعه ای از تراکنش ها و خطایابی برای دوباره به وجود آوردن یک حالت پایدار که بخشی از مکانیزم بازیافت اساسی (اصولی) است ، نیست. از آنجا که خطایابی قسمتی از سیستم است ،بنابراین احتمال آن زیاد است که خطایابی به صورت صحیح انجام شود. بعلاوه چون خود برنامه کوچکتر است،احتمال آن هم خیلی زیاد است که درست باشد. اعتقاد ما بر این است که با پشتیبانی این نوع از تبدیل ها (تغییر شکل ها)در سیستم اساسی ،نرم افزار ساخته خواهد شد آسانتر برای نوشتن و کمتر به شکست متمایل خواهد شد.

به هر حال همیشه تعداد کمی از برنامه های کاربردی وجود دارند که برای آنها نوشتن یک pager بر حسب عادت ضروری است ، بازیافت خوش بینانه قادر به پشتیبانی همه ی برنامه های کاربردی نخواهد بود.

بازیافت خوش بینانه همچنین به روز رسانی روی سایت های چندگانه را به صورت اتوماتیک انجام نمی دهد.

به هر حال،از آنجا که بازیافت خوش بینانه کنترل اضافی انجام نمی دهد، به صورت اتوماتیک بودن ،معمولاً نتیجه نمی شود.

ما بررسی می کنیم پی آمدهای (نتایج) کنترل اضافی را و اینکه چگونه مشکلات بازیافت و کنترل اضافی می تواند مستقلانه (آزادانه) در یک حالتی که هر یک از این دو یا هر دو ترکیب شده باشند، حل شود.

از آنجا که همه ی کارهای ما بر اساس پیگیری روابط (وابستگیها) است، ما می توانیم از یک مجموعه ی یکنواخت از نگهبانان محاسبه ای برای پیگیری (بررسی)مسندات گوناگون استفاده کنیم.

این به ما اجازه می دهد لغو کنیم نتیجه ی محاسبات اشتباه شده از خرابی پردازه را، کنترل اضافی یا محاسبات دیگر به سادگی با به عقب برگشتن برای رسیدن به یک حالت پایداری(همسان) دو طرفه حالت سیستم که هیچکدام از حالات ما شکست نخورده اند (نقض نشده اند) . این، مشکل ترکیب کردن تبدیل های چندگانه را در حد زیادی ساده سازی خواهد کرد.

ما ادامه می دهیم به بررسی مشکلات بازیافت ، کنترل اضافی ، تکرار و.... برای کار روی یک سیستمی که تمام این تبدیلات بر حسب نیاز موارد کاربری خاص،می تواند باهم بکار برده شود.

قسمت دوم از این نوشته یک مدل عمومی برای استدلال درباره ی این روشهای بازیافت در سیستمهای توزیع شده معرفی می کند. با این مدل نشان می دهیم که همیشه یک حالت سیستم قابل بازیافت ماکزیمم وجود دارد

که هرگز کاهش نمی یابد.

2.یک مدل

2.1 .حالتهای پردازه ها

هر لحظه که یک پردازه یک پیام ورودی دریافت می کند ، یک وقفه ی حالت جدید حالتهای پردازه را آغاز می کند،ترتیب قطعی اجرا فقط بر حالتی از پردازه در زمانی که پیام می رسد و روی محتویات خود پیام استوار است . داخل هر پردازه،هر وقفه ی حالت با یک اندیس وقفه ی حالت ترتیبی واحد شناسایی می شود که بسادگی یک شمارش تعداد پیام های ورودی که پردازه دریافت کرده است می باشد.تمام روابط (وابستگی های) پردازه ی i روی بعضی پردازه j می تواند کدگذاری شود بسادگی همچون اندیس ماکزیمم هر وقفه ی حالت پردازه ی j که پردازه ی i با آن رابطه دارد (به آن وابسته است). این کدگذاری ممکن است چون اجرای پردازه داخل هر وقفه ی حالت قطعی است و چون هر وقفه ی حالت در یک پردازه مسلماً وابسته است به وقفه های قبلی همان پردازه.

تمام روابط هر پردازه ی i با یک بردار وابستگی نشان داده می شود:

$$\mathrm{d}_i = \langle \delta_* \rangle = \langle \delta_1, \delta_2, \delta_3, \ldots, \delta_n \rangle \ ,$$

در اینجا n شماره ی کل پردازه در سیستم است . مؤلفه ی j از بردار وابستگی پردازه ی i ، $\delta_{j\_}$ ، اندیس ماکزیمم هر وقفه ی حالت پردازه ی j ای که پردازه ی i معمولاً به آن وابسته است را می دهد.

مؤلفه ی i بردار وابستگی خود پردازه ی i همیشه برحسب وقفه ی حالت جاری پردازه ی i مقداردهی می شود. اگر پردازه ی i هیچ وابستگی ای به وقفه ی حالت برخی پردازه های دیگر مثل j نداشته باشد، $\delta_{j\_}$ به صورت $\perp$ تنظیم می شود که از تمام اندیس های وقفه ی حالت ممکن کمتر است.

باهم کار کردن(همیاری کردن) برای حفظ (ادامه دادن) بردار وقفه ی حالت آنها با برچسب زدن تمام پیام های فرستاده شده با اندیس وقفه ی حالت جاری آنها و با یادآوری اندیس ماکزیمم در هر پیام دریافت شده از پردازه های دیگر در هر پردازه می باشد. در طول هر اجرای واحد از سیستم ،بردار وابستگی برای هر پردازه به طور منحصربفرد بوسیله ی اندیس وقفه ی حالت پردازه تعیین می شود. هیچ مؤلفه از بردار وابستگی هر پردازه میان اجرای نرمال برنامه نمی تواند کاهش پیدا کند.

2.2.حالتهای سیستم

حالت سیستم ترکیبی است ازحالت های تمام مؤلفه های پردازه های سیستم و می توانند با یک ماتریس وابستگی $n \times n$ نشان داده شوند:

$$\mathbf{D} = [\delta_{**}] = \begin{bmatrix} \delta_{11} & \delta_{12} & \delta_{13} & \dots & \delta_{1n} \\ \delta_{21} & \delta_{22} & \delta_{23} & \dots & \delta_{2n} \\ \delta_{31} & \delta_{32} & \delta_{33} & \dots & \delta_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \delta_{n1} & \delta_{n2} & \delta_{n3} & \dots & \delta_{nn} \end{bmatrix}$$

در این ماتریس : سطر i ، δ_{ij} و $1 \leq j \leq n$ در بردار وابستگی برای پردازه ی i است.

چون مؤلفه ی i از بردار وابستگی پردازه ی i همیشه اندیس وقفه ی حالت جاری پردازه ی i است، قطر ماتریس وابستگی، $\delta_{ii},\ 1 \leq i \leq n$ اندیس وقفه ی حالت جاری هر پردازه در سیستم را نشان می دهد.

اجازه بدهید $\mathcal{S}$ مجموعه ای از تمام حالتهایی که در طول هر اجرای واحد از سیستم اتفاق می افتند باشد که در آن یک حالت سیستم الویت می یابد بر دیگری اگر و تنها اگر در طول اجرا در ابتدا روی داده باشد.این می تواند بیان شود در شرایط اندیس وقفه ی حالت هر پردازه همانطور که در ماتریس وابستگی نشان داده شد.

تعریف ***1***

اگر $\mathbf{A} = [\alpha_{**}]$ و $\mathbf{B} = [\beta_{**}]$ حالت های سیستم در $\mathcal{S}$ باشند ، پس

$$\mathbf{A} \preceq \mathbf{B} \iff \forall i \left[\alpha_{ii} \leq \beta_{ii}\right]$$

این ترتیب جزئی متفاوت است از تعریف شده با Lamport's *happened* چون آن مرتب می کند حالتهای سیستمی که نتیجه ی رویدادها از خود آنها بیشتر است و فقط وقفه های حالت (شروع شده با دریافت پیام) رویدادها را تشکیل می دهد.

برای مثال ، شکل 1 یک سیستم چهار پردازه ی مختلف را نشان می دهد.خط افقی اجرای هر پردازه را بیان می کند، هر بردار یک پیام از یک پردازه به پردازه ی دیگر را نشان می دهد.

دو حالت ممکن $\mathbf{A}$ و $\mathbf{B}$ را ملاحظه کنید، که درحالت $\mathbf{A}$ ، پیام a دریافت شده است اما پیام b دریافت نشده است و در حالت $\mathbf{B}$ ، پیام b دریافت شده، اما پیام a دریافت نشده است.

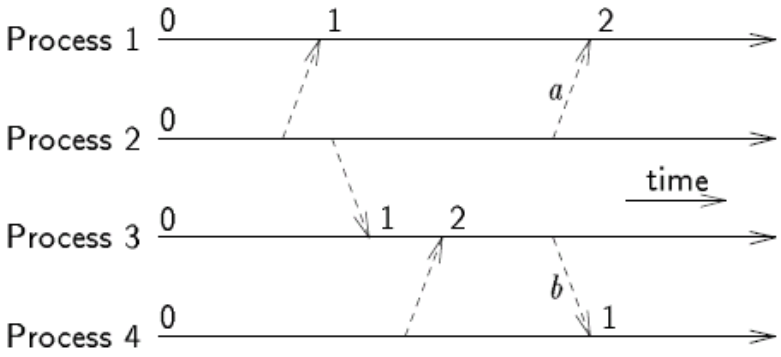


Figure 1 The system history partial order

این حالتها می توانند با ماتریس وابستگی بیان شوند:

$$\mathbf{A} = \begin{bmatrix} \textcircled{2} & 0 & \bot & \bot \\ \bot & 0 & \bot & \bot \\ \bot & 0 & 2 & 0 \\ \bot & \bot & \bot & \textcircled{0} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \textcircled{1} & 0 & \bot & \bot \\ \bot & 0 & \bot & \bot \\ \bot & 0 & 2 & 0 \\ \bot & \bot & 2 & \textcircled{1} \end{bmatrix}$$

حالت $\mathbf{A}$ و $\mathbf{B}$ مقایسه ناپذیرند تحت روابط قبلی سیستم،که این هم می تواند دیده شود با مقایسه کردن مقدارهای دایره کشیده شده روی قطرهای این دو ماتریس وابستگی.

2.3.شبکه بندی قبلی سیستم

یک حالت سیستم یک مجموعه از پیام هایی که دریافت شده اند با هر پردازه را شرح می دهد.دو حالت سیستم در $\mathcal{S}$ ترکیب می شوند برای تشکیل اجتماع آنها به طوری که پردازه ای تمام پیام ها را دریافت کرده است که آن یکی از دو حالت های سیستم اولیه را دارد. این می تواند بیان شده باشد در شرایط ماتریس های وابستگی شرح دهنده ی این

حالت های سیستم با انتخاب سطری برای پردازه که بزرگترین اندیس وقفه ی حالت سطر های متناظر در ماتریس های اولیه را دارد.

تعریف 2

اگر $\mathbf{A} = [\alpha_{**}]$ و $\mathbf{B} = [\beta_{**}]$ حالت های سیستم در $\mathcal{S}$ باشند ،اجتماع $\mathbf{A}$ و $\mathbf{B}$ برابر است با :

$$\mathbf{A} \cup \mathbf{B} = [\gamma_{**}],$$

همچنین اشتراک دو حالت سیستم در شکل می گیرد به طوری که هر پردازه دریافت کرده باشد فقط آن پیام هایی را که در هر دو حالت سیستم اولیه وجود دارند.

این می تواند شکل بگیرد از ماتریس های وابستگی شرح دهنده ی این حالت ها با انتخاب سطری که کوچکترین اندیس وقفه ی حالت در بین سطر های متناظر در ماتریس های اولیه را دارد .

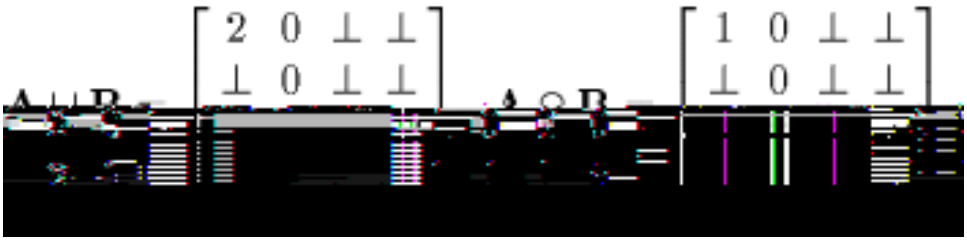
*تعریف **3***

اگر $\mathbf{A} = [\alpha_{**}]$ و $\mathbf{B} = [\beta_{**}]$ حالت های سیستم در $\mathcal{S}$ باشند ، اشتراک $\mathbf{A}$ و $\mathbf{B}$ برابر است با :

$$\mathbf{A} \cap \mathbf{B} = [\delta_{**}],$$

$$\forall\, i \left[\delta_{i*} = \left\{ \begin{array}{ll} \alpha_{i*} & \text{if } \alpha_{ii} \leq \beta_{ii} \\ \beta_{i*} & \text{otherwise} \end{array} \right. \right] .$$

در ادامه ی مثال قسمت 2.2 در شکل 1، اجتماع و اشتراک حالت های $\mathbf{A}$ و $\mathbf{B}$ می تواند شکل بگیرد با انتخاب سطر های مربوط از این دو ماتریس



قضیه ی زیر شبکه بندی قبلی سیستم تشکیل یافته به وسیله ی مجموعه ای از حالت های سیستم که در طول هر اجرای واحد از سیستم اتفاق می افتد را نشان می دهد که با رابطه ی قبلی سیستم مرتب شده اند.

قضیه 1

مجموعه ای که با رابطه ی پیشین سیستم مرتب شده ،یک شبکه تشکیل می دهد. برای هر $\mathbf{A}, \mathbf{B} \in \mathcal{S}$ کوچکترین کران بالای $\mathbf{A}$ و $\mathbf{B}$ هست $\mathbf{A} \cup \mathbf{B}$ وبزرگترین کران پایین $\mathbf{A}$ و $\mathbf{B}$ هست $\mathbf{A} \cap \mathbf{B}$.

اثبات

با توجه به ساختار اجتماع و اشتراک حالت سیستم در تعاریف 2 و3 این قضیه درست است.