ارائه روشی جهت بهبود درهم سازی کوکو مبتنی بر چگالی با استفاده از حذف عمل درهم سازی مجدد

پیمان عاربی [*,۱]

۱- گروه مهندسی کامپیوتر، دانشکده مهندسی برق و کامپیوتر ، دانشگاه فنی و حرفه ای، تهران، ایران

چکیده

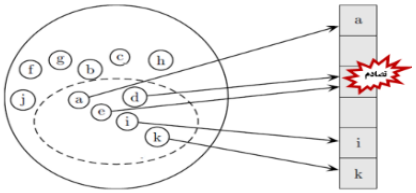
درهم سازی کوکو یکی از انواع روش های درهم سازی است که سعی دارد تا حد ممکن تعداد برخوردها را کاهش داده و عملیات در درهم سازی را با پیچیدگی ثابت انجام دهد. یکی از مهمترین مشکلات که در روش درهمسازی کوکو وجود دارد حلقه های بی نهایت در زمان اضافه کردن کلیدهاست که منجر به عمل درهمسازی مجدد می شود. تاکنون گونه های مختلفی برای بهبود این روش درهم سازی ارائه شده است که هرکدام به نحوی سعی کرده اند این مشکل را حل کنند. در این مقاله ابتدا گونه های مختلف درهم سازی کوکو با هم مقایسه شده است و سپس با ارائه پیشنهاداتی در جهت تغییر روش درهم سازی کوکو سعی شده است تا عمل درهمسازی مجدد از این روش حذف شود. این کار موجب شده است تا عمل اضافه کردن و میزان سربار حافظه بهبود یابد.

کلمات کلیدی: درهم سازی کوکو، زنجیره سازی، گراف کوکو، درهم سازی نامتقارن،تابع درهم سازی

۱. مقدمه

در علم کامپیوتر جداول درهم ساز و دیکشنری ها ساختمان داده های پرکاربردی هستند که با استفاده از آنها کلیدهای داده ایی درون یک جدول نگاشت می شود. ایده کلی بر این اساس است که به جای درنظر گرفتن خود کلیدها برای آدرس دهی از تابعی از کلیدها استفاده و هر داده را در اندیس متناظر با مقدار آن تابع ذخیره کنیم. هدف اصلی استفاده از جداول درهم سازی این است که سه عمل اضافه، حذف و جستجوی کلیدها در زمان $\Theta(1)$ انجام شده و در عین حال دسترسی به داده ها مستقیم باشد. اگر برد تابع درهم سازی از مرتبه تعداد داده ها باشد و این تابع روی کلیدهای مورد نظر به صورت یک به یک عمل کند، تمامی اطلاعات را می توان در آرایه ای به اندازه داده ها ذخیره کرد. اما اگر دو کلید متفاوت به یک اندیس مشابه نگاشته شوند پدیده ایی به نام "برخورد" بروز می کند(شکل۱). تابع درهم سازی ایده آل است که هیچ برخوردی را ایجاد نکند ولی این امر تقریبا غیرممکن است. حال باید با روشی سعی کنیم مسئله برخورد را حل کنیم.

* **Corresponding author: Peyman Arebi**
Email: parebi@tvu.ac.ir



شکل(۱). برخورد دو کلید در جدول درهم سازی

برای حل مشکل برخورد تاکنون روشهایی پیشنهاد شده است که مشهورترین آنها کلاس روش ها دو روش زنجیره سازی و آدرس دهی باز می باشد. در روش زنجیره ایی، کلیدهایی که با هم برخورد دارند را در یک لینک لیست به ترتیب بروز برخورد ذخیره می کنیم. در این حالت در بدترین حالت دسترسی به داده از مرتبه $\Theta(n)$ خواهد بود. از مهمترین الگوریتم های ارائه شده در این کلاس می توان به الگوریتم Chained Hashing و Two-Way Hashing اشاره کرد. در روش آدرس دهی باز هر بار که برخوردی صورت می گیرد اولین خانه خالی بعد از مکان برخورد جستجو شده و داده جدید در آنجا قرار می گیرد. الگوریتم Linear Probing مهمترین الگوریتم شناخته شده در این گروه می باشد. این روش نیز در پیاده سازی استاندارد در بدترین حالت دارای زمان $\Theta(n)$ در جستجوی کلیدهاست.

روش پرکاربرد دیگری که در زمینه کاهش برخورد در جداول درهم سازی ارائه شده است روشی است تحت عنوان درهم سازی کوکو که توسط راموس پاگ و فلیمینگ رادلر در سال ۲۰۰۴ ارائه شده است[1]. هدف اصلی این روش آن است که زمان جستجوی و هزینه سرشکنی بروزرسانی داده ها در بدترین حالت با زمان ثابت انجام شود. چالش های مختلفی در اجرای روش کوکو وجود دارد که تاکنون برخی از محققان روش هایی جهت برطرف نمودن این چالشها ارائه داده اند.

در این مقاله در بخش ۲ درهم سازی کوکو استاندارد معرفی شده است. سپس در بخش ۳ گونه های مختلفی از درهم سازی کوکو که هر کدام سعی کرده اند به نحوی این روش را بهبود دهند شرح داده و با هم مقایسه شده است. در بخش ۴ روش جدید را جهت بهبود این نوع درهم سازی ارائه داده و در بخش ۵ روش پیشنهادی را پیاده سازی شده است و درنهایت در بخش ۶ ضمن بررسی نتایج، روش پیشنهادی با درهم سازی استاندارد کوکو مقایسه شده است.

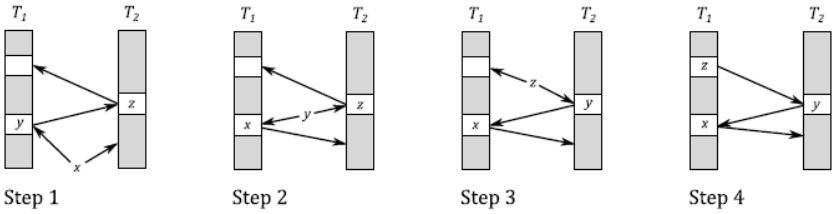
۲. درهم سازی کوکو

کلیه مقالات کامل

همانطور که گفته شد در عمل درهم سازی قرار است n عنصر در یک جدول به طول m ذخیره شود. در نسخه استاندارد درهم سازی کوکو از دو جدول(T1 و T2) برای درهم سازی استفاده می شود که طول هر یک $m = (1+\varepsilon)n$ می باشد؛ جاییکه $\varepsilon > 0$ است و مقداری ثابت دارد. به هر خانه از جداول درهم سازی یک باکت گفته می شود که در اینجا در هر باکت حداکثر یک آیتم قرار می گیرد. هر کدام از جداول دارای یک تابع درهم سازی(h1 و h2) مربوط به خود هستند که آیتم ها در هنگام ورود به جداول با استفاده از تابع مختص آن جدول کدگذاری می شوند. سه عملیات مهم در کوکو به صورت زیر است:

- **اضافه کردن**

در هنگام اضافه کردن آیتم x، ابتدا مقدار درهم سازی آن در جدول T1 (مقدار T1[h1(x)]) محاسبه شده و در صورتیکه این مکان خالی باشد، آیتم x در آن باکت قرار می گیرد؛ در غیر اینصورت(یعنی باکت قبلا توسط آیتمی مانند y پر شده باشد) آیتم x به جای y قرار داده شده و مقدار درهم سازی مربوط به جدول T2 برای آیتم y (مقدار T2([h2(y)]) محاسبه شده و آیتم y به جدول دوم منتقل می شود. در صورتیکه در جدول این مکان خالی باشد که y را قرار می دهیم در غیر اینصورت همانند قبل y در آن خانه قرار گرفته و آیتم جاری به جدول T1 منتقل می شود(شکل۲). این فرآیند تا زمانی ادامه می یابد که باکت خالی پیدا شود و آیتم نهایی در آن قرار گیرد. هزینه سرشکنی این عملیات $\theta(1)$ است.



شکل(۲). اضافه کردن آیتم x با روش درهم سازی کوکو

- **حذف کردن**

حذف در جداول درهم سازی کوکو به سادگی انجام می شود. ابتدا مقادیر درهمسازی h1 و h2 را برای آیتم مورد نظر حساب کرده و با دو مقایسه آیتم را که ممکن است در یکی از جداول باشد پیدا می کنیم و سپس باکت مربوط به آیتم را خالی می کنیم. این کار در زمان ثابت انجام می شود.

- **جستجو**

همانطور که در عمل حذف نیز گفته شد در زمان ثابت می توان با دو مقایسه آیتم مورد نظر را یافت و یا عدم وجود آیتم را اعلام کرد.

چالش های اساسی در فرآیندهای مختلف درهم سازی کوکو وجود دارد که محققان سعی کرده اند با ارائه روش هایی بر این مشکلات فائق آیند. چالشهای موجود در درهم سازی کوکو را می توان به دو دسته تقسیم کرد:

۱) چالش کمبود حافظه

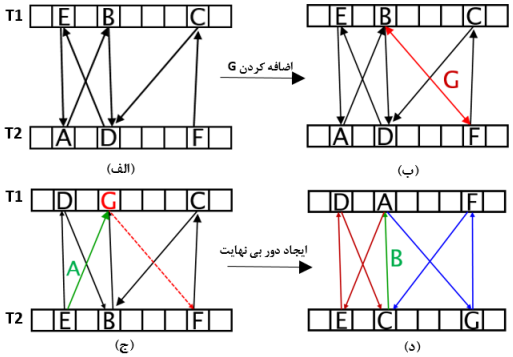
از آنجاییکه دو جدول با طول محدود m وجود دارد ممکن است این جداول پر شده و جایی برای اضافه کردن آیتم های جدید نباشد.

۲) چالش طولانی شدن زنجیره جابه جایی و ایجاد دور بی نهایت

همانطور که گفته شد در اضافه کردن آیتمها ممکن است عناصر بین دو جدول چندین بار جابه جا شوند. اگر این زنجیره جابه جایی زیاد باشد به تبع موجب می شود تا هزینه اضافه کردن آیتمها به شدت بالا رود.

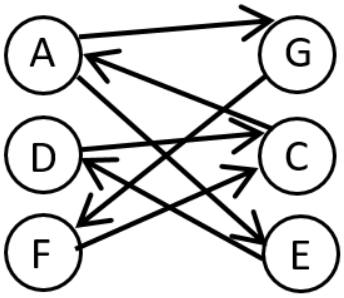
از طرفی پدیده دیگر در زمان اضافه کردن آیتمها ممکن است رخ دهد و آن ایجاد دور است. به عنوان مثال فرض کنید تعدادی کلید تا کنون به جداول کوکو اضافه شده است(شکل ۳ (الف)). برای درک بهتر در اشکال، مکان آیتم اضافه شده در جدول دیگر را با یک پیکان( ) نشان داده ایم به نحوی که مبدا پیکان مکانی از جداول است که آیتم اضافه شده و انتهای پیکان به باکتی اشاره می کند که همان مکان آیتم مورد نظر در جدول دیگر است. حال می خواهیم آیتم G را اضافه کنیم. مکان آیتم G در جدول T1 همان مکان آیتم B است(شکل ۳(ب)). بنابراین به صورت زنجیروار G با B، B با D، D با E

و E با A جابه جا می شود. وضعیت جداول تاکنون را در شکل ۳ (ج) مشاهده می کنید. حال آیتم A به دلیل اینکه با E جایگزین شده به جدول T1 انتقال داده می شود و بر اساس تابع درهم سازی مکان آن همان مکان G است. در این حالت دور ایجاد می شود یعنی به همان نقطه ای می رسیم که در ابتدای فرایند اضافه کردن(اضافه کردنG) بوده ایم. در این حالت مشکلی ایجاد نمی شود چون A با G جایگزین شده و G به جدول T2 رانده می شود. در ادامه آیتم G با F، F با ، C با B و همانطور که در شکل ۳(د) مشاهده می کنید B با A جایگزین خواهد شد. در این لحظه دور دوم اتفاق می افتد. درحالتی که دور دوم به وجود می آید فرآیند در یک حلقه بی نهایت قرار می گیرد.



شکل(۳). ایجاد دور بی نهایت در فرآیند درهم سازی کوکو

برای تحلیل بهتر این مشکل می توان روند اضافه کردن آیتم ها را به صورت یک گراف دو بخشی به نحوی نشان داد که در آن بخش ها همان جداول T1 و T2 و گره ها نیز باکتهای جداول باشند و یالها به صورت جهت دار و مطابق آنچه در شکل(۳) قرارداد شد بین گره ها برقرار شود. به این ساختار کوکو گراف گفته می شود. در شکل ۴ گراف مربوط به شکل۳(د) را مشاهده می کنید.



شکل(۴). گراف کوکو شکل ۳(د)

با رسم گراف به راحتی می توان دورها را شناسایی کرد. در شکل ۴ دور اول با مسیرA,E,D,C,A و دور دوم با مسیر A,G,F,C,A قابل رویت می باشد.

در زمانیکه فرآیند دچار حلقه بی نهایت می شود برای برون رفت از این بن بست می بایست عملیات درهم سازی مجدد(Rehashing) انجام شود؛ بدین صورت که اندازه جدول دوبرابر شده و تمامی داده ها از ابتدا درهم سازی می شوند. بدیهی است عمل درهم سازی مجدد هم از نظر حافظه و هم زمان پرهزینه بوده و زمان عملیات اضافه کردن را به O(n) افزایش می دهد.

۳. گونه های مختلف درهم سازی کوکو

بر اســاس ســه گروه از چالش هایی که در بخش قبل برشــمردیم، تلاش هایی در جهت رفع هر ســه گروه از چالش هاصورت گرفته شده است. خروجی این تلاش ها منجر به ایجاد گونه های جدید درهم سازی کوکو شده است که در ادامه مهمترین گونه های درهم سازی کوکو را معرفی و با هم مقایسه می کنیم.

- **درهم سازی کوکو نامتقارن**

درهم سازی کوکو استاندارد دارای لود غیر متوازن است؛ بدین دلیل که همواره ابتدا آیتم ها به جدول T1 اضافه شده و درصورتیکه باکت مربوطه اشغال باشد به جدول T2 منتقل می شوند. این باعث می شود همواره T1چگال تر از T2 باشد. از طرفی اندازه دو جدول T1 وT2 نیز با هم برابر اســت. بنابراین در گونه ای از درهم ســازی کوکو تحت عنوان درهم ســازی کوکو نامتقارن که توسط پاگ و رادلر پیشــنهاد شــده اســت، اندازه جدول T1 بیشــتر از جدول T2 انتخاب می شــود[1]. در نتیجه اندازه دو جدول نامتقارن می شــود. با توجه به اینکه میزان لود در T1 بیشــتر اســت افزایش اندازه T1 نســبت به T2 باعث می شود تا فضای بیشتری برای ذخیره کلیدها فراهم شود و از جابه جایی کلیدها کاسته شده و در پی ان حلقه بی نهایت کمتر اتفاق بیافتد. در این روش فاکتور نامتقارن(c) تعریف می شــود که بر اســاس آن اندازه دو جدول به صــورت زیر تعریف می شود.(مقدار c بین صفر و یک است)

$$m_1 = \lfloor m(1+c) \rfloor$$
$$m_2 = 2m - m_1$$

m: طول جداول در حالت استاندارد

نتایج نشان می دهد که هر چه قدر c به ۱ نزدیکتر باشد کارایی درهم سازی افزایش می یابد.

- **d-ary cuckoo hashing:**

زمانیکه تعداد کلیدها بسیار زیاد است معمولا به سرعت جداول T1 و T2 پر خواهد شد. در روش d-ary که در سال ۲۰۰۵ توسط فوتاکیس پیشنهاد شده است، برای درهم سازی از بیش از دو جدول استفاده شده است[2]. در این الگوریتم d جدول و d تابع درهم ســاز را در فرآیند درهم ســازی کلیدها به کار برده اســت. نقطه قوت این الگوریتم آن اســت که از فضای بیشتری برای ذخیره سازی کلیدها بهره می برد. نتایج نشان می دهد در صورتیکه در این روش d=۳ باشد فاکتور لود بیشینه(معیار کارایی عملیات درهم سازی: نسبت تعداد کلیدها جا شده در جدول درهم سازی به تعداد کل خانه های جدول) تا ۹۱٪ افزایش می یابد. به همین ترتیب در d=۴ این میزان به ۹۷٪، در d=۵ به ۹۹٪ افزایش خواهد یافت. در مقاله اصلی این روش دو گونه متفاوت از این الگوریتم معرفی شده است. در گونه اول همانند درهم سازی کوکو استاندارد، الگوریتم ابتدا از جدول T1 شروع به درج می کند ولی در گونه دوم از این الگوریتم می توان از هر یک از d جدول به صورت تصادفی عمل اضافه کردن را آغاز کرد. به دلیل افزایش تعداد باکتها برای تحلیل عملیات و شناسایی حلقه از یک ابرگراف اســتفاده می شــود[3]. با بررســی حلقه ها در این ابرگراف می توان روند اجرای فرآیند درهم ســازی را به دقت زیر نظر گرفت[4]. نتایج تحلیل بیشتر در مورد روش d-ary را می توانید در [6] مشاهده کنید.

- **درهم سازی کوکو ساده شده**

گونه دیگری از درهم سازی کوکو که توسط مبتکر آن یعنی پا گ و رادلر ارائه شده است بدین صورت است که به جای اســتفاده از دو جدول در فرآیند درهم ســازی با اندازه m از یک جدول با اندازه ۲m اســتفاده شــده اســت. در این حالت با ساتفاده از دو تابع درهم ساز، کلیدها در سراسر این جدول آدرس دهی می شود. در برخی از تحقیقات به این روش درهم سازی کوکو ساده شده اطلاق شده است. نتیجه ناشی از این تغییر موجب می شود تا احتمال خالی بودن باکت آدرس دهی شده توسط تابع درهم ساز اول(h1) افزایش یابد[7].

- درهم سازی کوکو با کمک حافظه کمکی یا Stash

همانطور که در بخش ۲ گفته شد، فرایند درهم سای کوکو ممکن است در یک حلقه بی نهایت گرفتار شود و عملیات اضافه شدن دچار خطا شده و منجر به دوباره سازی جداول درهم سازی شویم؛ که این خود بسیار پرهزینه است. برای غلبه بر این نقطه ضعف، کیریش و همکاران در سال ۲۰۰۸ روشی را ارائه داده اند که از یک حافظه اضافی که به آن stash گفته می شود استفاده کرده تا کلیدهایی که امکان جایگزین شدن را ندارند موقتا در این حافظه ذخیره شوند[8]. در حالت ساده این حافظه می تواند یک آرایه به صورت یک صف باشد، اما در صورتیکه تعداد آیتم های دچار مشکل زیاد باشد ممکن است این حافظه از جنس جداول درهم سازی باشد. کلیدهای قرار گرفته شده در این حافظه در فرصتهای بعدی پس از تغییرات احتمالی در جداول درهم سازی مجددا در باکتها قرار می گیرند. از آنجاییکه حافظه stash به وفور مورد دسترسی قرار می گیرد و عموما اندازه کمی دارد، آن را در حافظه کش قرار میدهند تا سرعت دسترسی افزایش یافته و وجود چنین حافظه ای کارایی الگوریتم را کاهش ندهد. گونه های متفاوتی از این روش ارائه شده است که هر کدام سعی داشته اند تا با کمک این حافظه از بروز حلقه بی نهایت و دوباره سازی جداول جلوگیری کرده و یا به تاخیر بیاندازند.

- **Cuckoo++ Hashing:**

این روش که توسط اسکارنک در سال ۲۰۱۷ به منظور افزایش کارایی شبکه های کامپیوتری پیشنهاد شده است سعی دارد تا با استفاده بیشتر از جدول درهمسازی T2 بتواند بار زیادی که بر روی جدول T1 است را تاحد زیادی کاهش دهد[9]. همچنین بر اساس ذات سیستمهای مبتنی بر شبکه از یک تایمر برای محدود کردن طول زنجیره جابه جایی آیتم ها و جلوگیری از ایجاد حلقه بی نهایت استفاده می کند.

برخی گونه های ارائه شده از درهم سازی کوکو با استفاده از روشهای پردازش موازی و همروند سعی دارند تا با بهبود معماری الگوریتم زمان اجرای درهم سازی را بهبود دهند.

همانطور که در تمامی روش ها مشاهده می کنید همگی عملیات اضافه کردن را مورد هدف قرار داده اند. این بدان دلیل است که اضافه کردن تنها عملی در ساختمان داده درهم سازی کوکو است که ممکن است تحت شرایی در بدترین حالت از مرتبه ایی غیر ثابت(O(n)) شود. در بین روش های ارائه شده معمولا روش هایی مفیدتر بوده است که توانسته است با کمترین هزینه از اجرای دوباره سازی(Rehashing) جلوگیری کند؛ که این خود محصول بروز حلقه های بی نهایت است.

۴. روش پیشنهادی برای بهبود درهم سازی کوکو

پرهزینه ترین عملیات در فرآیند درهم سازی کوکو عمل اضافه کردن است. همانطو که گفته شد ممکن است در نسخه استاندارد درهم سازی کوکو این عمل در بدترین حالت O(n) شود. دلیل این مشکل ایجاد زنجیره طولانی از جابه جایی و و بروز حلقه بی نهایت است که منجر به دوباره سازی جداول در همسازی می شود. بنابراین برای بهبود این ساختمان داده می بایست تغییراتی را در عملیات اضافه کردن انجام دهیم که هدف آن از یک طرف کاهش طول زنجیره جابه جایی هاست و از طرف دیگر از بروز حلقه بی نهایت جلوگیری کرده و یا به تاخیر بیاندازد. ما در اینجا در راستای این اهداف دو تغییر را در فرآیند اضافه کردن درهم سازی کوکو پیشنهاد می دهیم.

۱) تغییر (۱):

در عملیات استاندارد اضافه کردن کوکو آیتم اضافه شده ابتدا بدون هیچ بررسی به جدول T1 اضافه می

شد. درصورتیکه باکت مورد نظر اشغال بود، آیتم را در آن خانه قرار می داد و آیتم قبلی را جابه جا می کرد.

ما در اینجا این شیوه را به صورت دیگری انجام می دهیم، به نحوی که در زمان اضافه شدن آیتم x مقدار h1(x) و h2(x) رامحاسبه کرده و درصورتیکه T[h1(x)] خالی بود آیتم x را در آن باکت قرار می دهیم، در غیر اینصورت(یعنی T[h1(x)] پر بود) آیتم x را در باکت T[h2(x)] قرار می دهیم. در صورتیکه هر دو باکت پر بود برای شروع جابه جایی باکتی را انتخاب می کنیم که جدول مربوط به آن دارای چگالی بیشتر باشد. چگالی جدول درهم ساز iام را به صورت زیر محاسبه می کنیم:

$$\rho_i = \frac{A_i}{m_i} \qquad (۱)$$

که در آن

A: تعداد خانه های اشغال شده جدول iام قبل از اضافه شدن آیتم x است.

m: اندازه جدول درهم سازی iام

تغییرات در جهت توزیع مناسب کلید ها در جداول و کاهش چگالی جداول صورت گرفته شده است که این خود موجب کاهش طول زنجیره جابه جایی ها و بروز حلقه بی نهایت می شود.

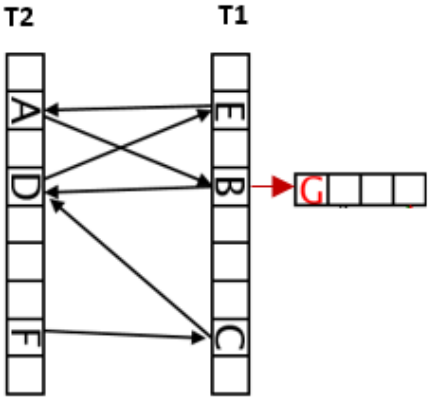
۲) تغییر (۲):

همانطور که گفته شد در دو صورت عمل درهم سازی مجدد انجام می شود. اولا زنجیره طولانی از جابه جایی ها و ثانیا بروز حلقه بی نهایت. برای جلوگیری از درهم سازی مجدد قبل از اضافه کردن آیتم x به جداول درهم ساز و شروع جابه جایی ها، ابتدا بر اساس گراف کوکو سناریو اضافه کردن آیتم x را در گراف کوکو بررسی کرده و در صورت شناسایی زنجیره جابه جایی طولانی(بیشتر از مقدار مرزی تعیین شده) و یا بروز حلقه بی نهایت در اثر اضافه شدن آیتم x، بدین صورت عمل می کنیم که در جدول T1 و در باکت مربوط به آیتم x زنجیره ایی همانند روش chaining ایجاد کرده و عنصر x در آن زنجیره قرار می گیرد. این زنجیره از جنس لیست پیوندی است. در حقیقت در اینجا در زمان بروز زنجیره جابه جایی طولانی و یا حلقه بی نهایت از مکانیزمی استفاده می شود که در روش chaining بکار گرفته می شد.

با اعمال این تغییر تعداد اجرای عمل درهم سازی مجدد(Rehashing) را به صفر می رسانیم. با حذف عمل درهم سازی مجدد هزینه حافظه و اجرای الگوریتم به شدت کاهش خواهد یافت. از طرفی اضافه شدن زنجیره به ساختمان داده درهم سازی سرباری ایجاد خواهد کرد اما معمولا در درهم سازی کوکو و به خصوص با اعمال تغییر(۱) تعداد آیتم هایی که منجر به حلقه بی نهایت می شوند به شدت کاهش می یابد و وجود این زنجیره ها سربار جزئی را به الگوریتم وارد کرده و هزینه سرشکنی عمل اضافه کردن را در زمان ثابت نگه می دارد.

دقت داشته باشید که فقط آیتم هایی در زنجیره لیست پیوندی قرار می گیرند که اضافه کردن آنها منجر به حلقه بی نهایت و درهم سازی مجدد می شود.

به عنوان مثال در لحظه اضافه شدنG در مثال شکل ۳(د) بر اساس تغییر(۲) ساختار درهم سازی به شکل (۵) خواهد شد.



شکل(۵). اعمال تغییر(۲) در فرآیند درهم سازی کوکو

با توجه به ایجاد زنجیره ها ممکن است زمان اجرای عمل جستجو تحت تاثیر قرار بگیرد، اما به دلیل تعداد کم عناصر در زنجیره ها، هزینه سرشکنی عمل جستجو برای n عنصر در همان مقدار ثابت باقی می ماند.

۵. پیاده سازی روش پیشنهادی

حال برای بررسی میزان بهبود روش پیشنهادی نسبت به الگوریتم استاندارد درهم سازی کوکو در اینجا تغییرات پیشنهادی را بر روی نسخه استاندارد کوکو اعمال می کنیم. برای پیاده سازی روش پیشنهادی از نسخه پیاده سازی شده درهم سازی کوکو استاندارد دانشگاه استنفورد که توسط کیت شوارتز به زبان جاوا نوشته شده است استفاده کردیم[10] و تغییرات (۱) و (۲) را بر روی آن اعمال کردیم. در این پیاده سازی از دو فرم تابع درهم سازی مایکل ثراپ (برای H1,H2) که برای اولین بار در روش Linear Probing ارائه شد استفاده شده است.

در این آزمایش مقدار مرزی زنجیره جابه جایی را $0.2m$ درنظر گرفتیم و $\epsilon = 0.1$ و همچنین مقادیر پارامترهای Low و High در تابع درهم سازی مایکل به ترتیب ۱۶ و ۳۲ بیت درنظر گرفتیم.

برای ارزیابی روش پیشنهادی نسبت به نسخه استاندارد کوکو از سه معیار استفاده شده است:

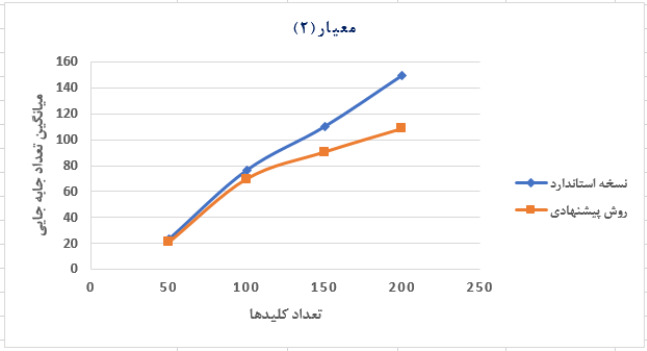
- معیار(۱): تعداد بروز حلقه بی نهایت (برای بررسی کارایی تغییر(۱))
- معیار(۲): تعداد جابه جایی آیتم ها بین جداول برای تمامی کلیدها
- معیار(۳): مدت زمان انتظار اضافه شدن کلیه کلیدها(مدت زمان انتظار برابر است با مدت زمانی از شروع اضافه شدن اولین کلید تا پایان اضافه شدن آخرین کلید طول می کشد)

تمامی معیارهای فوق را بر اساس تعداد کلیدها در اجراهای مختلف مورد سنجش قرار شده است. به همین منظور اجراهای مختلفی با تعداد کلیدهای ۵۰،۱۰۰،۱۵۰،۲۰۰ انجام شده است.

برای اینکه نتایج به دست آمده به ترتیب ورود کلیدها وابسته نباشد، این آزمایش را ۱۰ بار به ترتیب ورود تصادفی کلیدها انجام دادیم و معیارهای فوق را به صورت میانگین گزارش شده است.

در اشکال ۶ تا ۸ به ترتیب نمودارهای مربوط معیارهای ۲،۱ و۳ را نسبت به تعداد کلیدها در عمل اضافه کردن مشاهده می شود.

شکل(۶).نمودار میانگین تعداد حلقه های بی نهایت نسبت به تعداد کلیدها



شکل(۷).نمودار میانگین تعداد جابه جایی ها نسبت به تعداد کلیدها

شکل(۸).نمودار زمان انتظار نسبت به تعداد کلیدها

با توجه به اینکه در روش پیشنهادی حافظه جدیدی در زمان زنجیره سازی اضافه می شود می بایست میزان سربار حافظه را محاسبه کرد. در روش استاندارد کوکو در لحظه درهم سازی مجدد، حافظه ایی به میزان $m \times \epsilon$ به جدول اضافه می شود اما در روش پیشنهادی در زمان ایجاد حلقه دوم به اندازه یک گره حافظه اضافه می شود. در شکل(۹) نمودار سرباز اضافه شده در روش پیشنهادی با روش استاندارد در اجراهای با تعداد کلیدهای مختلف مشاهده می کنید. در این نمودار واحد میزان سربار حافظه یک کلمه حافظه درنظر گرفته شده است.

شکل(۹): نمودار سربار حافظه

۶. تعریف متغیّرها بحث در مورد نتایج

نتایج بدست آمده در نمودار شکل(۶) نشان می دهد که تعداد حلقه های بی نهایت در زمانیکه تعداد کلیدها کم است تفاوت چندانی بین روش استاندارد و پیشنهادی نداشته است. اما با افزایش کلیدها تعداد حلقه ها به دلیل مکانیزمی که در تغییر (۱) اعمال کردیم به شدت کاهش یافته است. این نشان می دهد که تغییرات اعمال شده در (۱) توانسته است تا حد زیادی از بروز حلقه ها جلوگیری کند که این خود موجب کاهش طول زنجیره های لیست پیوندی در روش پیشنهادی می شود و سربار اضافه شده را به شدت کاهش می دهد. از طرف دیگر تعداد جابه جایی ها در روش پیشنهادی نسبت به روش استاندارد در اندازه های مختلف کلیدها کاهش چشمگیری را نشان می دهد که این خود مهمترین نقطه قوت روش پیشنهادی می باشد. این کاهش در تعداد کلیدهای بالا افزایش بیشتری داشته که این امر کمک می کند تا کارایی روش پیشنهادی بسیار خوب باشد. مدت زمان انتظار برای اضافه کردن کلیدها در روش پیشنهادی به مراتب پایین تر از روش استاندارد است. میزان زمان انتظار در حالتی که تعداد کلیدها ۲۰۰ عدد است در روش پیشنهادی به یک سوم نسبت به روش استاندارد کاهش یافته است. این معیار نشان دهنده سرعت بالای اجرای الگوریتم پیشنهادی است. با بررسی میزان سربار حافظه در الگوریتم پیشنهادی نسبت به نسخه استاندارد، همانطور که قابل پیش بینی بود در روش پیشنهادی میزان سربار به شدت کاهش یافته است که این از برجسته ترین ویژگی های روش ارائه شده است و از این رو مناسب بسیستم هایی است که دارای محدودیت حافظه هستند. نقطه قوت دیگر روش پیشنهادی تسریع در عمل اضافه کردن نسبت به روش کوکو استاندارد است؛ به همین دلیل در سیستمهایی که پویا هستند یعنی داده ها به سرعت حذف و اضافه می شوند روش پیشنهادی کارایی بالایی دارد.

۱۲. مراجع

[1] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. Journal of Algorithms, 51(2):122–144, 2004.

[2] .Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. Theory Comput. Syst., 38(2):229–248, 2005.
[3] Reinhard Diestel. Graph Theory. Springer, Berlin, 2005.
[4] Tsiry Andriamampianina and Vlady Ravelomanana. Enumeration of connected uniform hypergraphs. In FPSAC 05, 2005.
[5] Jurek Czyzowicz, Wojciech Fraczak, and Feliks Welfed. Notes about d-cuckoo hashing. Technical Report RR 06/05-1, Universit´e du Qu´ebec en Outaouais, 2006.
[6] Michael Drmota and Reinhard Kutzelnigg. A precise analysis of cuckoo hashing. preprint, 2008.
[7] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. In Proceedings of the 16th Annual European Symposium on Algorithms, 2008.
[8] Nicolas Le Scouarnec, Cuckoo++ Hash Tables: High-Performance Hash Tables for Networking Applications ,Journal of Networking and Internet Architecture , 18(3):101–114, 2017
[9] http://www.keithschwarz.com/interesting/code/cuckoo-hashmap/CuckooHashMap.java.html