چای سبز:
معماری جدید زبان Go برای مدیریت بهینه‌ی حافظه

سید حسین موسوی فرد
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
hajmousa1385@gmail.com

۱۱ دقیقه

گو یک زبان برنامه‌نویسی سطح بالا است که توسط گوگل عرضه شده و بیشتر به‌علت سهولت استفاده و سرعت بالا مورد توجه قرار گرفته است. مدیریت حافظه در زبان‌های سطح پایین (مانند C)، بر عهدهٔ توسعه‌دهنده گذاشته شده است تا بیشترین سطح کنترل روی برنامه حاصل شود. اما در زبان‌های سطح بالا (مانند جاوا، پایتون و گو) برای ساده‌سازی فرایند توسعهٔ برنامه و همچنین جلوگیری از به وجود آمدن خطا در حافظه، از سامانه‌ای به نام «زباله‌جمع‌کن[۱]» استفاده می‌شود. همان‌گونه که انتظار می‌رود، وجود چنین سامانه‌ای موجب کُندی عملکرد برنامهٔ اصلی می‌شود. پس بهینه بودن زباله‌جمع‌کن حائز اهمیت است. تیم توسعهٔ زبان گو با ارائهٔ الگوریتمی جدید به بهبود فرایند پرداخته است. در این مقاله به بررسی ساختار زباله‌جمع‌کن‌ها به‌ویژه از نوع ردیاب می‌پردازیم و الگوریتم جدید چای سبز را به‌طور کلی توضیح می‌دهیم.

در علوم کامپیوتر، داده به هر نوع اطلاعاتی گفته می‌شود که توسط برنامه‌ها برای محاسبه، پردازش و تصمیم‌گیری به‌کار می‌رود. در واقع داده رکن اصلی برنامه تعریف می‌شود. وقتی اطلاعاتی به برنامه داده می‌شود، پردازش آن نیازمند ذخیره شدن داده در جایی از کامپیوتر است. شیوهٔ مدیریت داده‌ها در حافظه بر بازدهی، پایداری و امنیت برنامه مؤثر است. در زبان‌های سطح پایین مدیریت داده توسط برنامه‌نویس انجام می‌شود تا کنترل کامل روی برنامه وجود داشته باشد. راحتی در توسعه یکی از اهداف زبان‌های سطح بالا است و همچنین مدیریت دستی حافظه ممکن است منجر به خطاهایی مانند نشت حافظه یا اشاره‌گر معلق[۲] بشود. برای جلوگیری از این مشکلات، زباله‌جمع‌کن‌ها معرفی شدند. در یک تعریف کوتاه زباله‌جمع‌کن به سامانه‌ای خودکار گفته می‌شود که وظیفه‌اش آزاد کردن حافظه‌ای است که دیگر مورد استفادهٔ برنامه نیست. به این بخش از حافظه که دیگر توسط برنامه استفاده نمی‌شود «زباله» گفته می‌شود.

شیوهٔ کار زباله‌جمع‌کن

الگوریتم‌های مختلفی برای سامانهٔ زباله‌جمع‌کن ارائه شده است و گو از میان آن‌ها، الگوریتم Mark-Sweep را، که نوعی الگوریتم ردیاب[۳] است، برگزیده است. این الگوریتم متشکل از دو مرحلهٔ نشانه‌گذاری(mark) و جارو کردن(sweep) می‌باشد. در لحظهٔ اجرا شدن زباله‌جمع‌کن، هر متغیر درون برنامه یک ریشه است و الگوریتم مرحلهٔ mark را شروع می‌کند. سامانه شروع به مسیریابی از ریشه‌ها می‌کند و هر شیئی را که به آن برسد، علامت‌گذاری می‌کند. سپس در مرحلهٔ جارو کردن، زباله‌جمع‌کن کل حافظه را اسکن می‌کند و اگر شیئی در آن وجود داشت که در مرحلهٔ اول علامت‌گذاری نشده بود، آن را آزاد[۴] می‌کند تا جا برای ایجاد اشیای جدید مهیا شود.

۱. Garbage collector
۲. Dangling pointer
۳. Tracing garbage collector
۴. Deallocate

بررسی دقیق‌تر

اشیای زباله به دو نوع تقسیم می‌شوند: دستوری[۵] و معنایی[۶]. وقتی متغیری در برنامه وجود داشته باشد که به شیئی اشاره کند اما آن متغیر هیچ‌وقت در برنامه استفاده نشود، شیء به یک زبالهٔ دستوری تبدیل می‌شود. از به وجود آمدن این نوع زباله، معمولاً در زمان کامپایل جلوگیری می‌شود. زبالهٔ معنایی به شیئی گفته می‌شود که برنامه پس از یک یا چند بار استفاده، دیگر با آن کاری ندارد و وجود آن‌ها در حافظه زائد است.

```go
package main

import (
    "fmt"
)

func main() {
    var x int
    var y int

    x = 12
    fmt.Println(x)

    fmt.Println(2+2) // X is now a semantic garbage
}
```

برای مثال، در کد بالا متغیر y یک زبالهٔ دستوری محسوب می‌شود چون در هیچ جایی از برنامه دسترسی به آن ایجاد نشده است. متغیر x پس از اجرای دستور چاپ، یک زبالهٔ معنایی است، زیرا در ادامهٔ روند برنامه به آن اشاره‌ای نشده است. حال نوبت به زباله‌جمع‌کن می‌رسد که فضای اشغال‌شده توسط زباله‌ها را برای برنامه آزاد کند.

در زبان گو، زباله‌جمع‌کن به صورت دوره‌ای و در بازه‌های زمانی مشخص اجرا نمی‌شود. بلکه go runtime هر وقت که یک متغیر جدید در حافظهٔ heap ایجاد می‌شود، میزان رشد استفاده از این حافظه توسط برنامه را محاسبه می‌کند. اگر این رشد از ۱۰۰ درصد بیشتر شده باشد (یعنی از آخرین بار که go runtime بررسی کرده، میزان استفاده از حافظه دو برابر شده باشد)، زباله‌جمع‌کن اجرا می‌شود تا فضای خالی برای برنامه ایجاد کند.

صفحه‌بندی، شیوهٔ ذخیرهٔ اشیا در حافظه

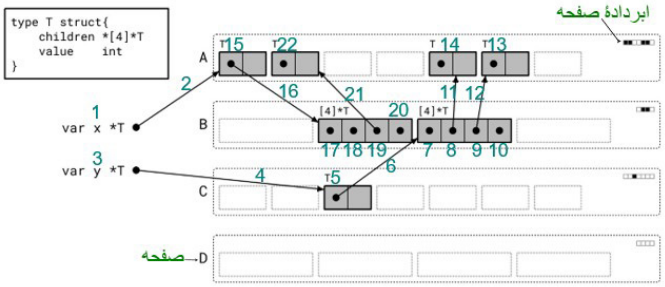
صفحه کوچک‌ترین واحد نقشه‌بندی حافظه در رم است که توسط سیستم‌عامل مدیریت می‌شود. گو برای گرفتن تکه‌های حافظه از سیستم‌عامل

۵. Syntactic
۶. Semantic

صفحه‌ها را درخواست می‌کند. اشیاء در زبان گو، از نظر اندازه (بر اساس نوع داده)، به چند کلاس تقسیم می‌شوند. هر صفحه شامل اشیایی با کلاس اندازهٔ یکسان است (برای مثال، اشاره‌گرها و ساختارهای(Struct) کوچک، در کلاس ۸ بایتی قرار می‌گیرند.).

مرحلهٔ اول – mark

زباله‌جمع‌کن از ریشه‌ها (متغیرهای درون فضا) شروع می‌کند و مکان آن‌ها در حافظه را علامت‌گذاری می‌کند. اگر یک متغیر در درون ساختار خود به شیء دیگری اشاره کند، سامانه به بررسی آن‌ها می‌پردازد و همهٔ اشیایی را که از طریق ریشه‌ها -چه مستقیم و چه غیر مستقیم- قابل دسترسی‌اند، اسکن می‌کند. اگر شیئی غیر از این موارد باقی بماند، به این معناست که برنامه نمی‌تواند به آن‌ها دسترسی داشته باشد. تصویر زیر مثالی از چگونگی انجام این مرحله است.



زباله‌جمع‌کن از ریشه‌ها (متغیرهای موجود) شروع می‌کند و به بررسی آن‌ها در حافظه می‌پردازد. سامانه برای هر صفحه، درون رجیسترهای پردازنده یک فهرست (ابردادهٔ درون تصویر) تعریف می‌کند. در این فهرست به ازای هر شیء درون صفحه، یک بیت در فهرست وجود دارد. با بررسی هر شیء، بیت مربوط به آن در فهرست به یک تبدیل می‌شود (به عنوان دیده‌شده علامت‌گذاری می‌شود). در انتها، اشیاء قابل دسترسی، آدرسی درون فهرست دارند. بیت‌هایی که درون فهرست برابر صفر هستند، متعلق به اشیای غیرقابل دسترس یا فضای خالی‌اند.

مرحلهٔ دوم – sweep

در این مرحله سامانه شروع به خواندن کل حافظه می‌کند و هر شیئی را که در فهرست اشیای قابل دسترسی نیست، آزاد می‌کند.

اشکالات

مانند همهٔ زباله‌جمع‌کن‌ها، به هنگام اجرا، روند اجرای برنامه متوقف می‌شود و زباله‌جمع‌کن شروع به کار می‌کند. این اجرا باعث ایجاد وقفه‌های دوره‌ای درون برنامه می‌شود و عملکرد آن را تحت‌تأثیر قرار می‌دهد. همچنین ممکن است در روند اجرای سامانه‌های بی‌درنگ[۷] اختلال ایجاد شود (ممکن است زمانی که برنامه باید نتیجه را تحویل بدهد، زباله‌جمع‌کن در حال اجرا و برنامه متوقف شده باشد). در نتیجه، بهینه بودن الگوریتم‌های جمع کردن زباله موجب افزایش کارایی برنامه می‌شود.

۷. Real-time systems

غیرقابل پیش‌بینی بودن سرعت دسترسی

پردازنده داده‌هایی را که از حافظه می‌خواند، کش می‌کند. اما برخی داده‌های موجود در حافظه که به یکدیگر اشاره می‌کنند لزوماً در یک صفحه یا نزدیک به هم نیستند و نمی‌توان به‌طور قطع گفت همهٔ اشیای مورد نیاز برای بررسی، درون کش وجود دارند. از این رو پردازنده مجبور است برخی داده‌ها را از حافظه فراخوانی کند که موجب کند شدن فرایند می‌شود. (استفاده از حافظه حدود ۱۰۰ برابر کندتر از کش است). همچنین حدود ۳۵ درصد از زمان مرحلهٔ مارک صرف انتظار برای دریافت داده‌ها از حافظه می‌شود (پردازنده منتظر است و جای دیگری به‌کار نمی‌رود).

ساختار جدید «دسترسی ناهمگن به حافظه[۸]»

این ساختار متناسب با نیازهای سرورها و سیستم‌های چندسوکتی (سیستم‌هایی که بیش از یک پردازنده دارند) طراحی شده است به‌طوری که هر حافظه، مخصوص دسترسی یکی از پردازنده‌ها قرار گرفته است و دسترسی هر پردازنده به حافظهٔ منتسب به خودش، نسبت به حافظهٔ دیگر پردازنده‌ها، تأخیر کمتری دارد. بر این اساس، سرعت دریافت اشیاء از حافظه برای زباله‌جمع‌کن به پردازندهٔ اجراکنندهٔ زباله‌جمع‌کن و موقعیت شیء مورد نظر درون حافظه وابسته است.

مشکل در پردازنده با تعداد هسته‌های بالا

همچنین هر هستهٔ پردازنده یک بخش(go routine) از زباله‌جمع‌کن را اجرا می‌کند. برای جلوگیری از تعارض‌هایی مانند پردازش دوبارهٔ یک شیء یا اصلاً پردازش نشدن آن، اشیاء یک صف کار[۹] تشکیل می‌دهند و به نوبت توسط زباله‌جمع‌کن پردازش می‌شود. این یعنی در سیستم‌هایی با تعداد هسته‌های بالا، تمام هسته‌ها در انتظار تکمیل کار زباله‌جمع‌کن می‌مانند و عملاً مقدار زیادی از توان پردازشی بی‌استفاده رها می‌شود.

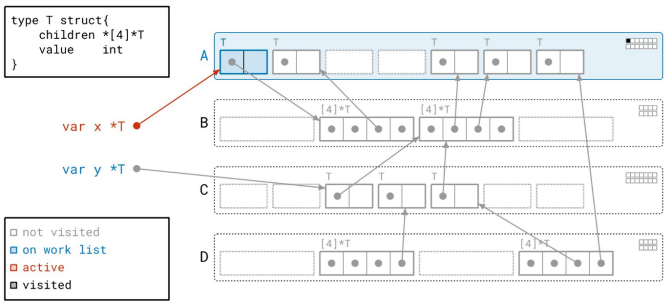
راه حل چای سبز

نکتهٔ کلیدی زباله‌جمع‌کن چای سبز، تمرکز بر روی صفحه‌ها می‌باشد (بر خلاف الگوریتم قبلی که بر اشیاء متمرکز بود). به جای اسکن کردن اشیاء در نوبت، صفحه‌ها به طور کامل اسکن می‌شوند (یعنی به جای قرار گرفتن اشیاء در صف پردازش، صفحه‌ها در صف قرار داده می‌شوند). مثل الگوریتم قبلی، فرایند از ریشه شروع می‌شود و سامانه، صفحهٔ حاوی شیئی را که ریشه به آن اشاره می‌کند، در صف پردازش قرار می‌دهد. شیء به عنوان دیده‌شده در فهرست علامت زده می‌شود. حال کار اسکن صفحه آغاز می‌شود: یک شیء وجود دارد که در فهرست دیده‌شده‌ها است، اما اسکن نشده است. سامانه آن را اسکن می‌کند تا اشاره‌گرهای درون آن را بیابد و اشیایی را که به آن‌ها اشاره شده است، در فهرست دیده‌شده‌ها بگنجاند (همچنین صفحهٔ حاوی شیء اشاره‌شده را در صف پردازش می‌گذارد). همین فرایند برای همهٔ ریشه‌ها تکرار می‌شود تا همهٔ آن‌ها بررسی شده باشند (برای جلوگیری از اسکن دوبارهٔ یک شیء، به ابردادهٔ صفحه یک فهرست «اسکن‌شده» اضافه شده است تا اگر شیئی دیده و اسکن شده است، به‌خاطر وجود در فهرست دیده‌شده‌ها، مجدداً اسکن نشود).

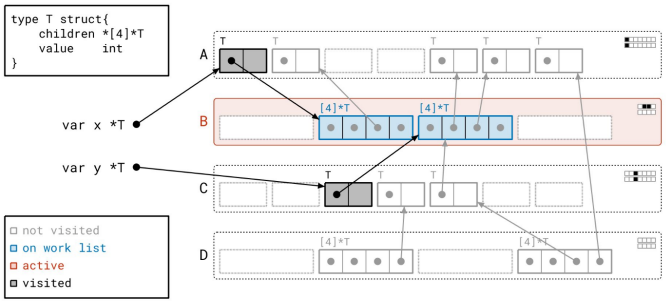
۸. Non-uniform memory access

۹. Work queue

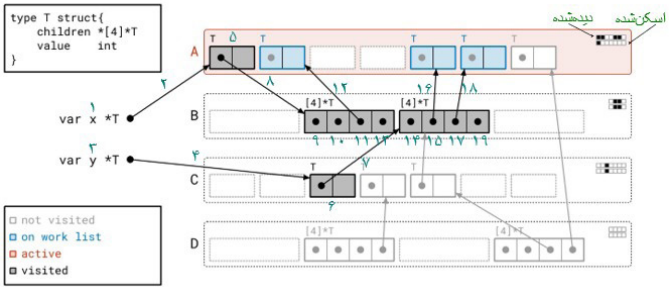
با مثال قبلی توضیح می‌دهیم:



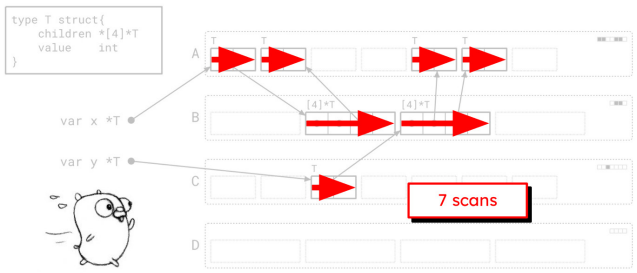
سامانه کار خود را از متغیر x آغاز می‌کند. شیئی را که x به آن اشاره می‌کند، در فهرست به عنوان دیده‌شده علامت می‌زند. سپس صفحهٔ حاوی شیء در صف پردازش قرار می‌گیرد. اکنون نوبت به پردازش می‌رسد. در صفحه، یک شیء وجود دارد که دیده شده ولی اسکن نشده است. سامانه آن را اسکن می‌کند و اشاره‌گری به یک شیء در صفحهٔ B پیدا می‌کند. شیء را در فهرست دیده‌شده‌ها قرار می‌دهد و صفحهٔ B را به صف اضافه می‌کند. همین اتفاق برای متغیر y نیز اجرا می‌شود.



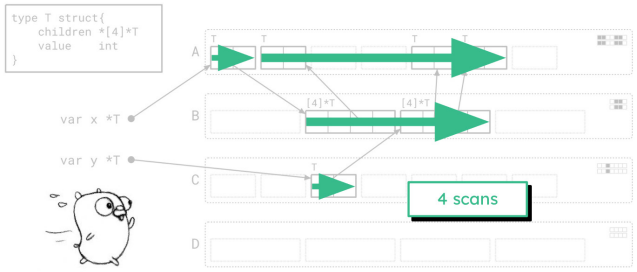
حالا فقط صفحهٔ B درون صف مانده است. زباله‌جمع‌کن اشیای دیده‌شده در صفحهٔ B را اسکن می‌کند و به سه شیء در صفحهٔ A می‌رسد. بنابراین، صفحهٔ B از صف حذف و صفحهٔ A به آن اضافه می‌گردد. این الگوریتم ادامه می‌یابد تا دیگر هیچ صفحه‌ای در صف نمانده باشد و همهٔ اشیایی که دیده‌شده‌اند، اسکن نیز شده باشند.



تفاوت بین دو الگوریتم را می‌توان به رانندگی در کوچه‌ها و رانندگی در بزرگراه تشبیه کرد. در الگوریتم قبلی، زباله‌جمع‌کن باید بین صفحه‌ها جابه‌جا شود که این موضوع خود زمان زیادی را صرف می‌کند. اما در روش جدید این جابه‌جایی به مقدار قابل توجهی کاهش یافته است.



الگوریتم قبلی: هر فلش نشانگر یک اسکن است.



الگوریتم جدید: تعداد اسکن‌ها از ۷ به ۴ کاهش یافته (چون اسکن بر اساس صفحه انجام می‌شود).

نتایج

بر اساس آزمایش‌های انجام‌شده، چای سبز زمانی بین ۱۰ تا ۴۰ درصد کمتر از زباله‌جمع‌کن قبلی مصرف می‌کند. همچنین در حال حاضر به عنوان زباله‌جمع‌کن پیش‌فرض در پروژه‌های گوگل استفاده می‌شود که نشان از بالغ شدن و آماده بودن آن دارد.

استفاده

زباله‌جمع‌کن چای سبز در نسخهٔ ۱.۲۵ گو به صورت آزمایشی افزوده می‌شود و در نسخهٔ ۱.۲۶ به زباله‌جمع‌کن پیش‌فرض تبدیل می‌شود. برای استفاده در نسخهٔ ۱.۲۵، می‌توان متغیر محیطی GOEXPERIMENT را روی greenteagc تنظیم کرد.

```
GOEXPERIMENT=greenteagc go run
```

منابع

[1] “The Green Tea Garbage Collector” in Michael Knyszek and Austin Clements blog:29 October 2025

[2] Paging in Operating System” t. GeeksforGeeks. Retrieved:2024-12-14.

[3] Red Hat. “Capacity Tuning”. Archived from the orgina:2017-07-23