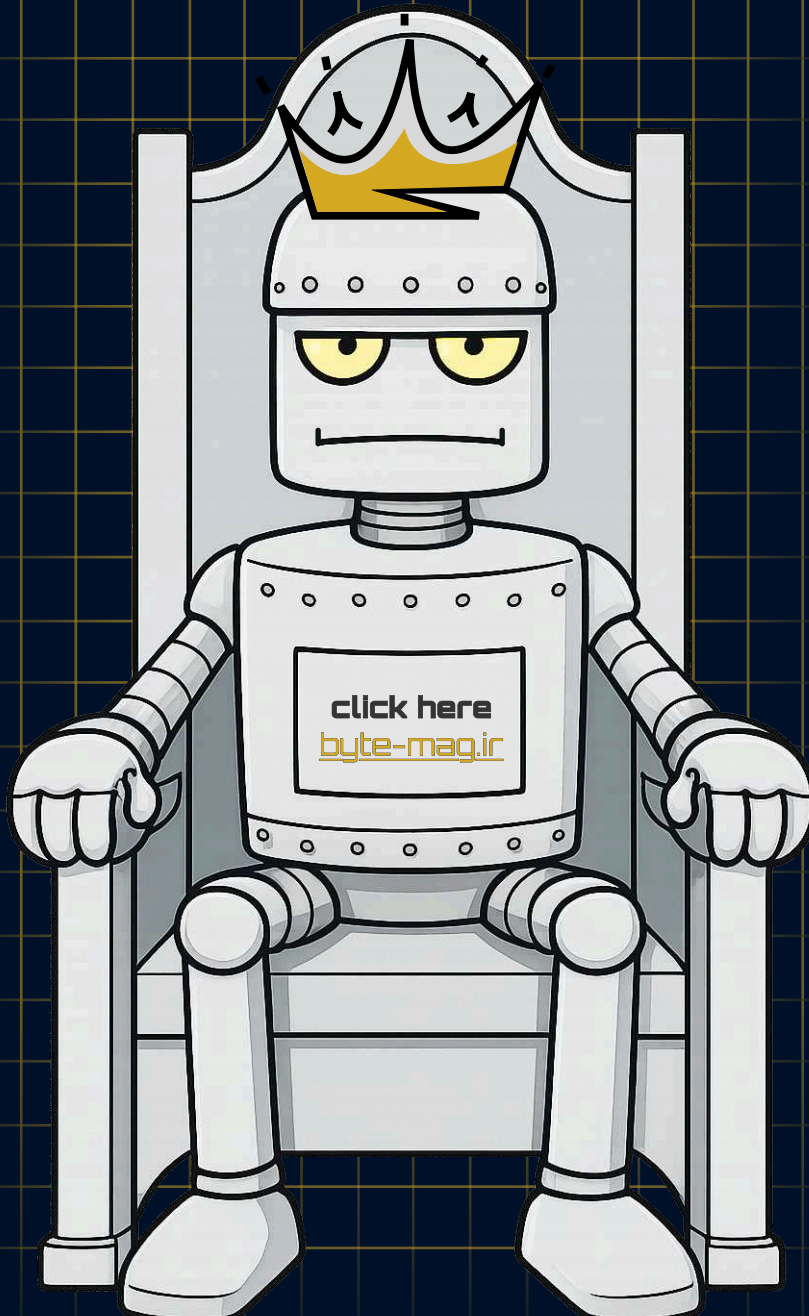


byte

شهریور ۱۴۰۴



فهرست

- آرش شاه حسینی ۲ دقیقه ۱
خطاها دروغ نمی گویند (سرمقاله) ۳
- امیرمهدی هدایتی پور ۵ دقیقه ۱
رایانش و ارتباطات کوانتومی؛ انقلابی در دنیای فناوری ۴
- عارف زارعزاده ۵ دقیقه ۱
عدد پی در توزیع نرمال ۶
- فرزام کوهی ۶ دقیقه ۱
کلید افزایش کارایی در پردازنده‌ها:
بافر مقصد انشعاب یا BTB ۸
- تحریریه ۲ دقیقه ۱
راوی آینده ۱۰
- محمد مهدی رجبی ۵ دقیقه ۱
سیستم‌های توزیع شده: هنر طراحی غیرممکن ۱۲
- مهدی بهرامیان ۴ دقیقه ۱
زبان Zig: فراتر از C با تمرکز بر کارایی و شفافیت کد ۱۴

صاحب امتیاز: انجمن علمی دانشکده مهندسی کامپیوتر (SSC)

مدیرمسئول: امیرحسین شهیدی

سردبیر: آرش شاه حسینی

صفحه آرایی و گرافیک: آرش شاه حسینی، امیررضا اینانلو و معین آعلی

ویراستاران ادبی: امیرحسین صوری، متین غیاثی، صهیب صادقی،

امیرحسین محمدزاده و امیرمهدی نامجو

تحریریه: آرمان طهماسبی، معین آعلی، مهدی علی نژاد، متین غیاثی و

امیرحسین صوری



خطاها دروغ نمی‌گویند!

سرمقاله

اما همین فریادها هستند که نقاب رویاهای ما را کنار می‌زنند. در رویای دیرینه‌مان همواره سیستمی بی‌نقص وجود دارد؛ سیستمی که هیچ‌گاه از مسیر خارج نشود و همه چیزش پیش‌بینی‌پذیر باشد. اما خطاها مدام به ما یادآوری می‌کنند که کنترل مطلق، توهمی بیش نیست.

اما آیا می‌توان به خطا نگاه دیگری داشت؟

در ادبیات، شکست گاهی از خود روایت عمیق‌تر است. سکوت یا جمله‌ای نیمه‌تمام می‌تواند بیش از هزار کلمه سخن بگوید.

پس چرا در مهندسی، خطا را فقط چیزی برای حذف و اصلاح بدانیم؟ شاید خطایک آغاز باشد؛ آغازی برای بازاندیشی در معنای درست کار کردن.

سیستمی که بی‌خطاست یا هرگز اجرا نشده، یا آن‌قدر ساده است که ارزش جدی گرفتن ندارد. انسانی هم که بی‌اشتباه است، یا زندگی نگرفته، یا هرگز قدم به ناشناخته‌ها نگذاشته است.

پس باید گفت زمان آن رسیده است که خطاها را نشانه‌های زندگی ببینیم؛ لحظه‌هایی که نشان می‌دهند هنوز در حال حرکت هستیم و هنوز چیزی فراتر از توان امروزمان می‌خواهیم.

و شاید همین نگاه است که شماره جدید بایت را خواندنی می‌کند؛ جایی که در میان فناوری، خطاها و پیچیدگی‌ها، هنوز امکان بازاندیشی و حرکت به سوی فردایی روشن وجود دارد.

در جهانی که شاخه علوم کامپیوتر بر پایه دقت، صحت و بهینه‌سازی بنا شده است، خطاها اغلب دشمن سیستم تلقی می‌شوند؛ پیام‌هایی مزاحم، که روند اجرا را متوقف می‌کنند و ما را وادار به اصلاح می‌سازند.

اما با نگاهی دقیق‌تر، خطاها نه تنها اجتناب‌ناپذیرند، بلکه نمودی از صادق‌ترین لحظه‌های یک سیستم هستند؛ کدهای تمیز و مستند در تلاش هستند تا همه چیز را بی‌نقص نشان دهند، اما خطاها چیزی را پنهان نمی‌کنند و دروغ نمی‌گویند. آن‌ها مثل آینه‌ای اند که واقعیت را بی‌پرده نشان می‌دهند و از این‌رو گاهی صاف و ساده‌تر از خود ما عمل می‌کنند.

درست همین موضوع است که ما را به یاد جهان فیزیکی می‌اندازد؛ جایی که باگ‌ها شبیه ترک‌های روی دیوارند؛ نشانه‌ای از فشار، زمان و واقعیت. ترک‌ها به ما یادآوری می‌کنند که هیچ سازه‌ای ابدی نیست. خطاها نیز به ما نشان می‌دهند که هیچ سیستمی کامل و بی‌نقص نیست.

نکته این‌جاست: فقط کدی که هرگز اجرا نشود، خطا هم نمی‌دهد. همان‌طور که انسانی که هرگز ریسک نکند، هرگز شکست را تجربه نمی‌کند.

پس هر پیام خطا سندی است بر اجرا شدن؛ فریادی کوچک است که می‌گوید: «این‌جا چیزی درست کار نمی‌کند و چه چیزی از این صادقانه‌تر؟»





در دنیای فناوری‌های نوظهور، مفاهیم کوانتومی از جمله مباحثی هستند که آینده محاسبات و ارتباطات را دگرگون خواهند کرد. این حوزه را می‌توان به‌طور کلی به چند شاخه اصلی تقسیم کرد:

۱. رایانش کوانتومی^۱

۲. ارتباطات کوانتومی^۲

۳. شبیه‌سازی کوانتومی^۳

۴. حسگرهای کوانتومی^۴

در این مقاله، تمرکز ما بر دو حوزه مهم رایانش کوانتومی و ارتباطات کوانتومی است. ابتدا با معرفی ارتباطات کوانتومی به‌عنوان شاخه‌ای مستقل که با هدف انتقال ایمن اطلاعات ایجاد شده، آغاز می‌کنیم و سپس سری به دنیای هیجان‌انگیز رایانش کوانتومی، دلایل نیاز به آن، الگوریتم‌های خاص کوانتومی و پیامدهای آن در رمزنگاری خواهیم زد. همچنین نگاهی به رمزنگاری پساکوانتومی (که راهکار مقابله با تهدیدهای رایانش کوانتومی در فضای امنیت سایبری است) خواهیم داشت.

ارتباطات کوانتومی: انتقالی ایمن و غیرقابل نفوذ

ارتباطات کوانتومی در نگاه اول ممکن است شبیه به اینترنت یا مخابرات کلاسیک به نظر برسد، اما ماهیت و ساختار آن کاملاً متفاوت است. این فناوری بر پایه اصول مکانیک کوانتومی، مثل «درهم‌تنیدگی» و «اصل عدم قطعیت هایزنبرگ» شکل گرفته است. در اینجا، اطلاعات به‌صورت کیوبیت^۵ منتقل می‌شوند؛ واحدی که برخلاف بیت کلاسیک، می‌تواند هم‌زمان در وضعیت صفر و یک باشد. اما جذاب‌ترین ویژگی ارتباطات کوانتومی، امنیت آن است. به لطف اصول کوانتومی مثل «درهم‌تنیدگی کوانتومی»، اگر فردی تلاش کند به اطلاعات دسترسی پیدا کند یا شنود انجام دهد، سیستم فوراً متوجه شده و ارتباط قطع شده یا تغییر می‌کند. همین ویژگی باعث شده رمزنگاری کوانتومی به‌عنوان امن‌ترین روش انتقال داده در نظر گرفته شود. ارتباطات کوانتومی بیشتر در شبکه‌های ایمن و توسعه اینترنت کوانتومی مطرح است، که از لحاظ اهداف، فناوری‌ها و کاربردها کاملاً از رایانش کوانتومی مجزا است.

رایانش کوانتومی: استفاده از اصول کوانتومی برای افزایش توان پردازشی

رایانش کوانتومی با هدف حل مسائل پیچیده‌ای توسعه یافته که رایانه‌های کلاسیک در آن‌ها ناکارآمد هستند. تفاوت اصلی رایانه کوانتومی با کلاسیک، در نحوه نمایش و پردازش اطلاعات است.

در رایانش کلاسیک، اطلاعات با استفاده از بیت‌ها نمایش داده می‌شوند که فقط می‌توانند در یکی از دو حالت صفر یا یک باشند. اما در رایانش کوانتومی، واحد اطلاعاتی به‌نام کیوبیت تعریف می‌شود. یک کیوبیت می‌تواند به‌طور هم‌زمان در ترکیبی از صفر و یک قرار گیرد. به این پدیده «برهم‌نهی» می‌گویند.

رایانه‌های کوانتومی با استفاده از روش‌های احتمالاتی می‌توانند برخی الگوریتم‌ها را سریع‌تر کنند. همین موضوع دلیل اصلی قدرت فوق‌العاده آن‌ها در حل برخی مسائل خاص است؛ مثلاً در تحلیل کلان‌داده‌ها، مدل‌سازی مولکولی یا بهینه‌سازی.

در حالی که ویژگی‌هایی مانند درهم‌تنیدگی نیز در طراحی و کارایی سیستم‌های کوانتومی نقش دارند؛ در بسیاری از الگوریتم‌های اولیه، به‌ویژه الگوریتم‌هایی مانند الگوریتم شور^۶، این برهم‌نهی است که به‌عنوان موتور اصلی محاسبه به‌کار می‌رود.

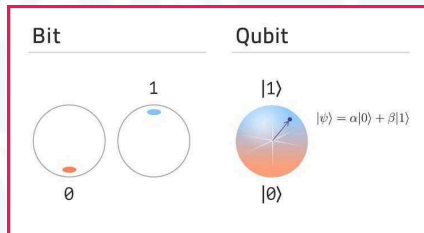
کیوبیت‌ها چه هستند؟

کیوبیت، مخفف «کوانتوم بیت»، واحد پایه اطلاعات در رایانش کوانتومی است. اما برخلاف بیت کلاسیک که فقط می‌تواند صفر یا یک باشد، یک کیوبیت می‌تواند هم‌زمان صفر و یک باشد. این حالت به‌صورت ریاضی با ترکیبی از دو بردار پایه نمایش داده می‌شود. که در آن، α و β ضرایب مختلط هستند و تعیین می‌کنند با چه احتمالی نتیجه صفر یا یک دیده می‌شود.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

در عمل، کیوبیت می‌تواند روی اجزای فیزیکی مختلفی پیاده‌سازی شود، مثل اسپین الکترون، قطبش فوتون، یا مدارهای ابررسانا. کنترل دقیق این کیوبیت‌ها و حفظ پایدار آن‌ها در برابر نویز محیطی، چالش بزرگ

مهندسی در ساخت رایانه‌های کوانتومی است. در واقع هر کیوبیت شبیه یک بیت احتمالاتی است که با یک احتمال صفر و با یک احتمال مقدار یک دارد و ما الگوریتم‌های خود را با پیاده‌سازی عملگرهای منطقی بر روی کیوبیت‌ها اجرا می‌کنیم.



تفاوت بیت و کیوبیت

چرا به رایانش کوانتومی نیاز داریم؟

در دهه‌های گذشته، پیشرفت رایانه‌ها بر اساس قانون مور^۷ پیش رفته است: هر دو سال، تعداد ترانزیستورها در یک تراشه دو برابر می‌شود. اما این روند به محدودیت‌های فیزیکی نزدیک شده است. نمی‌توان ترانزیستورها را بی‌نهایت کوچک کرد، چون در مقیاس نانومتر، اثرات کوانتومی خودبه‌خود پدیدار می‌شوند و اختلال ایجاد می‌کنند.

از سوی دیگر، برخی مسائل ذاتاً پیچیده هستند و حتی با قوی‌ترین رایانه‌های کلاسیک، حل آن‌ها زمان بسیار زیادی می‌برد. مثل شبیه‌سازی دقیق مولکول‌ها در شیمی کوانتومی یا حل مسائل بهینه‌سازی پیچیده. اما حل این مسائل به کمک رایانه‌های کوانتومی و استفاده از اصل برهم‌نهی در زمانی معقول ممکن است.

الگوریتم‌های کوانتومی و تاثیر آن‌ها بر رمزنگاری

یکی از مشهورترین الگوریتم‌های کوانتومی، الگوریتم شور است که می‌تواند عددهای بزرگ را به‌سرعت به عوامل اول تجزیه کند. این مسئله، برای رمزنگاری اهمیت زیادی دارد.



تهدید رایانش کوانتومی برای رمزنگاری کلاسیک

یکی از بزرگترین تهدیدهای رایانش کوانتومی متوجه حوزه رمزنگاری است، به ویژه رمزنگاری‌هایی مثل RSA که امروزه به شکل گسترده‌ای در اینترنت، بانک‌ها، امضای دیجیتال و بسیاری از سیستم‌های امنیتی استفاده می‌شوند.

الگوریتم RSA بر پایهٔ سختی تجزیهٔ عددهای بزرگ به عوامل اول ساخته شده است. به بیان ساده، اگر دو عدد اول بزرگ را در هم ضرب کنیم، به دست آوردن حاصل آن بسیار ساده است؛ اما اگر فقط حاصل ضرب را داشته باشیم، بازگشت به آن دو عدد اول (یعنی تجزیه) با رایانه‌های کلاسیک به شدت زمان‌بر و دشوار است. زمان‌بر بودن تجزیهٔ اعداد، همان مسئله‌ای است که RSA را ایمن می‌کند.

در رایانش کلاسیک، سریع‌ترین الگوریتم‌های تجزیهٔ اعداد در بهترین حالت دارای پیچیدگی زیرنمایی بر حسب اندازهٔ عدد ورودی هستند (دقت کنید n در این جا اندازهٔ عدد ورودی است، یعنی مثلاً تعداد بیت‌های آن). این یعنی زمان لازم برای شکستن کلید با اندازهٔ ۲۰۴۸ بیت، ممکن است هزاران سال طول بکشد.

اما رایانه‌های کوانتومی با بهره‌گیری از الگوریتم شور می‌توانند همین کار را با پیچیدگی زمانی چند جمله‌ای انجام دهند.

دانستنی

یکی از بزرگ‌ترین تهدیدهایی که با ظهور رایانش کوانتومی مطرح شده، حملهٔ موسوم به «ذخیره کن و بعداً رمزگشایی کن» است. مهاجم امروزه داده‌های رمزنگاری شدهٔ حساس (مثلاً تراکنش‌های مالی، مکاتبات دولتی و به خصوص تراکنش‌های رمز ارزها) را جمع‌آوری و ذخیره می‌کند، حتی اگر فعلاً نتواند آن‌ها را رمزگشایی کند. در آینده، وقتی رایانه‌های کوانتومی قدرتمند به طور کامل عملیاتی شوند و بتوانند الگوریتم‌هایی مثل شور را اجرا کنند، مهاجم می‌تواند به سرعت این داده‌های ذخیره شده را رمزگشایی کند. به این ترتیب، اطلاعاتی که سال‌ها ایمن به نظر می‌رسیدند، ناگهان افشا خواهند شد. زمانی که کامپیوترهای کوانتومی بتوانند سیستم‌های رمزنگاری امروزی را بشکنند، کیو دی^۱ یا آخرت کوانتومی رخ خواهد داد.

این حمله به ویژه در حوزه‌هایی مثل امنیت ملی و امنیت دارایی‌های دیجیتال بسیار خطرناک است. به طور خاص در حوزهٔ رمز ارزها مهاجمان از هم‌اکنون تراکنش‌های رمزنگاری شده را ذخیره می‌کنند تا در آینده، با کمک سخت‌افزارهای کوانتومی بتوانند به کلیدهای خصوصی و دارایی‌های موجود در آن‌ها دست یابند. به همین دلیل، حتی اگر رایانه‌های کوانتومی در حال حاضر به طور گسترده در دسترس نباشند، مهاجمان انگیزهٔ کافی برای ذخیرهٔ داده‌ها را دارند. این مسئله سازمان‌ها را مجبور می‌کند که از همین امروز به سمت الگوریتم‌های مقاوم در برابر کوانتوم (رمزنگاری پساکوانتومی) حرکت کنند تا از افشای آینده جلوگیری کنند.

به طور خاص، برای عددی به طول n بیت، در زمان $O(n^3)$ قابل اجراست (در برخی پیاده‌سازی‌ها حتی سریع‌تر). این یعنی برخلاف الگوریتم کلاسیک، بزرگترین کلیدها در زمانی معقول (مثلاً چند دقیقه) گشوده خواهند شد. این اختلاف زمانی عظیم باعث می‌شود که ECC، RSA و بسیاری از رمزنگاری‌های رایج دیگر در برابر حملات کوانتومی آسیب‌پذیر شوند. حتی سیستم‌هایی که امروز ایمن هستند، ممکن است در آینده با ضبط و ذخیره‌سازی داده و تحلیل آن در عصر کوانتومی، دچار نشت اطلاعات شوند (ذخیره‌سازی و رمزگشایی در آینده - store now, decrypt later).

رمزنگاری پساکوانتومی؛ راهی مستقل از تجزیه

برای مقابله با تهدیدهای کوانتومی، شاخه‌ای به نام «رمزنگاری پساکوانتومی» به وجود آمده است. در این روش ما به دنبال روش‌های رمزنگاری‌ای هستیم که برخلاف استانداردهای موجود به تجزیهٔ اعداد متکی نباشند. رمزنگاری‌هایی که حتی با الگوریتم‌های کوانتومی هم در زمان معقول گشوده نشوند.

برخی از مهم‌ترین رویکردها در این حوزه عبارتند از:

- رمزنگاری مبتنی بر شبکه‌ها
- رمزنگاری مبتنی بر چند جمله‌ای چندمتغیره
- رمزنگاری مبتنی بر کدهای تصحیح خطا

این الگوریتم‌ها از نظر محاسباتی، در برابر رایانه‌های کلاسیک و کوانتومی مقاوم طراحی شده‌اند. در همین راستا، سازمان‌هایی مانند NIST (سازمان ملی استاندارد و تکنولوژی ایالات متحده آمریکا) در حال تدوین و استانداردسازی مجموعه‌ای از الگوریتم‌های پساکوانتومی برای جایگزینی ECC و RSA هستند.

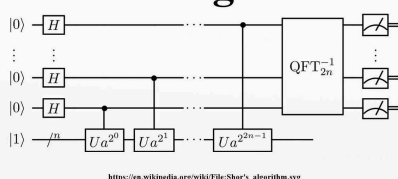
با این‌که رمزنگاری پساکوانتومی راهکاری موقت برای تأمین امنیت ارتباطات خواهد بود، اما با توجه به نوظهور بودن الگوریتم‌های کوانتومی هنوز با قطعیت مشخص نیست که آیا واقعاً کامپیوترهای کوانتومی از رمزگشایی این نوع رمزنگاری‌ها در زمان معقول عاجز هستند یا نه. به بیان دیگر ممکن است به زودی برای رمزگشایی برخی از این روش‌ها الگوریتم‌های کوانتومی یافت شوند.

حرف آخر؛ رمزنگاری؛ تنها یک برگ از دفتر کوانتوم

رایانش کوانتومی با استفاده از اصول مکانیک کوانتومی به ما در حل مسائلی که تا پیش از این غیرقابل حل به نظر می‌رسیدند کمک می‌کند. حل مسائل هم‌ارز با تجزیهٔ اعداد و رمزگشایی از کلیدهای RSA، تنها بخش کوچکی از ماجراست. رایانش کوانتومی احتمالاً حوزه‌هایی مثل یادگیری ماشین و هوش مصنوعی را متحول خواهد کرد و با توجه به نوظهور بودن رایانش کوانتومی مشخص نیست که دیگر چه حوزه‌هایی را درگیر خود خواهد کرد.

چه بخواهیم، چه نه، دوران رایانش کلاسیک به نقطهٔ اشباع نزدیک شده و رایانش کوانتومی در حال ورود به صحنه است. از یک طرف، رایانه‌های کوانتومی قدرت پردازش بی‌سابقه‌ای دارند و از طرف دیگر، ارتباطات کوانتومی راهی تازه برای انتقال امن اطلاعات فراهم می‌کند. هرچند هنوز چالش‌های مهندسی زیادی باقی مانده، اما آینده، بی‌شک به رنگ کوانتوم خواهد بود.

Shor's algorithm



الگوریتم شور



فرض کنید در حال صحبت با شخصی که خیلی ریاضی بلد نیست، هستید؛ به او درمورد قانون اعداد بزرگ و توزیع نرمال توضیح داده و به این که توزیع نرمال می تواند توصیف کننده خیلی چیزها در طبیعت باشد اشاره می کنید. مثلاً به او می گوئید که قد افراد در یک منطقه جغرافیایی از توزیع نرمال پیروی می کند. این موضوع برای آن شخص باید خیلی عجیب باشد؛ توزیعی که اکثر چیزهای طبیعی را توصیف می کند، باورکردنی به نظر می آید. سپس فرض کنید به آن شخص فرمول تابع چگالی احتمال (PDF) توزیع نرمال را نشان داده و اجزایش را توضیح می دهید:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

وقتی آن شخص، عدد π که نسبت محیط دایره به قطرش است را در این فرمول می بیند، قطعاً فکر می کند که دارید سربه سرش می گذارید! توزیع نرمال چه ربطی به محیط دایره دارد؟

این که عدد پی در توزیع نرمال وجود دارد چیزی است که در ابتدا احتمالاً برای ما عجیب بوده و بعد از مدتی بدون گرفتن دلیل قانع کننده ای این موضوع را پذیرفته ایم. اگر هم دنبال دلیل وجودش رفته ایم، احتمالاً به ما گفته اند دلیلش این است که انتگرال e^{-x^2} روی $\mathbb{R}$ برابر است با $\sqrt{\pi}$ و به صورت ریاضیاتی آن را نشان داده اند.

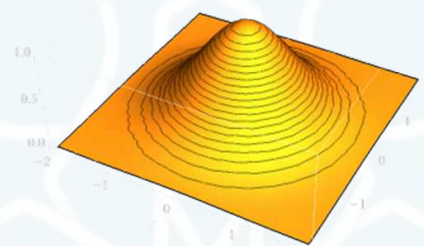
اما این توضیحات شاید برای بعضی قانع کننده نباشد یا شهود لازم برای درک این موضوع را فراهم نکند. اگر بخواهیم روی این موضوع دقیق شویم، باید سه سوال را جواب بدهیم تا این موضوع برای شخصی که نخستین بار است این موضوع را می بیند، قانع کننده باشد:

- ۱- انتگرال e^{-x^2} چه ارتباطی با دایره و محیطش دارد؟
 - ۲- اصلاً چرا این که e^{-x^2} با دایره ارتباط دارد طبیعی است؟
 - ۳- چرا قانون اعداد بزرگ ما را به تابع نمایی e^{-x^2} می رساند؟
- در این متن سعی می کنیم به دو سؤال اول جواب بدهیم.

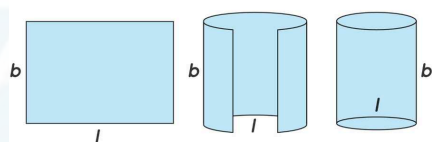
انتگرال e^{-x^2} را چگونه می توان حساب کرد؟ در ریاضی خوانده ایم که انتگرال و مشتق به نوعی برعکس هم دیگر هستند، پس اولین راهی که برای محاسبه آن ممکن است به ذهن مان برسد، پیدا کردن تابعی است که مشتقش e^{-x^2} باشد. اما مشکلی که وجود دارد، این است که انتگرال e^{-x^2} جواب تحلیلی ندارد. یعنی نمی توان آن را بر حسب توابعی که می شناسیم، مانند توابع مثلثاتی، لگاریتم و... بیان کرد. پس سوالی که پیش می آید این است که اصلاً چگونه می توان $\int_{-\infty}^{\infty} e^{-x^2} dx$ را حساب کرد؟ برای این کار، تکنیکی استفاده می کنیم که برای کسانی که اولین بار است آن را می بینند، خیلی عجیب به نظر می رسد؛ انتگرال را به یک بعد بالاتر می بریم. یعنی سعی می کنیم $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)} dx dy$ را حساب کنیم. واضح است که داریم:

$$\begin{aligned} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)} dx dy &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-x^2} e^{-y^2} dx dy \\ &= \int_{-\infty}^{\infty} e^{-x^2} \left(\int_{-\infty}^{\infty} e^{-y^2} dy \right) dx = \left(\int_{-\infty}^{\infty} e^{-y^2} dy \right) \left(\int_{-\infty}^{\infty} e^{-x^2} dx \right) \\ &= \left(\int_{-\infty}^{\infty} e^{-x^2} dx \right)^2 \end{aligned}$$

پس اگر $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)} dx dy$ را حساب کنیم، با یک رادیکال گرفتن می توانیم به خواسته سوال برسیم. اگر ریاضی ۲ را به یاد داشته باشید، می توانید خیلی راحت با تغییر متغیر به دستگاه مختصات قطبی این انتگرال را حساب کنید. ما هم از یک روش مشابه برای به دست آوردن جواب این انتگرال استفاده می کنیم، اما بدون استفاده از تکنیک های انتگرال دوگانه و کاملاً به صورت هندسی و شهودی. برای محاسبه $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)} dx dy$ اول باید بفهمیم چه چیزی است. چون انتگرال را هم در راستای x و هم در راستای y روی کل $\mathbb{R}$ می گیریم و به این معنی است که می خواهیم کل حجم زیر نمودار $e^{-(x^2+y^2)}$ را با این انتگرال محاسبه کنیم. نمودار این تابع به این شکل است:



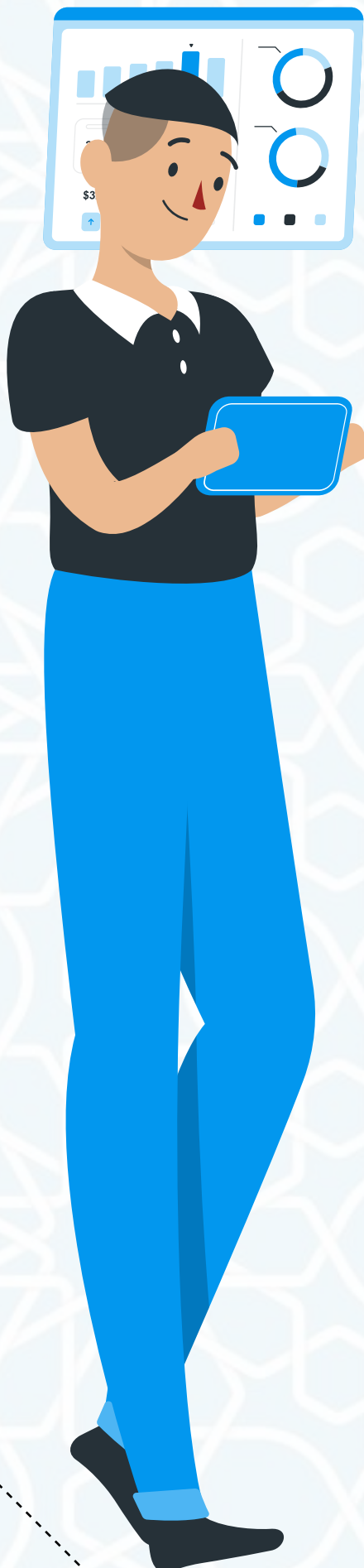
اگر دقت کنیم، می بینیم نقاطی که در راستای x و y فاصله یکسانی از مبدأ دارند، ارتفاع یکسانی هم دارند. این منطقی است، چون اگر فاصله از مبدأ را r در نظر بگیریم، طبق قضیه فیثاغورس داریم $r^2 = x^2 + y^2$ ، پس ارتفاع آن نقاط برابر است با e^{-r^2} . پس این نقاط ارتفاع یکسانی دارند و در نتیجه، یک استوانه را تشکیل می دهند. حال، برای اینکه حجم زیر این سطح را حساب کنیم، یک پوسته خیلی نازک استوانه ای (به ضخامت Δr) و به مرکز محور z در نظر می گیریم و سعی می کنیم حجم آن را به دست بیاوریم. فرض می کنیم شعاع داخلی این استوانه r هست، پس ارتفاعش در بازه e^{-r^2} و $e^{-(r+\Delta r)^2}$ قرار می گیرد. ما می توانیم ارتفاعش را ثابت گرفته و فرض کنیم مساوی با e^{-r^2} است. این فرض ما با کوچک تر شدن Δr دقیق تر می شود، پس در حد $\Delta r \rightarrow 0$ این فرض ما دقیق است (این حرف ما از لحاظ ریاضیاتی دقیق نیست، اما چون هدف ما داشتن شهود به این مسئله است، آن را درست در نظر می گیریم). همچنین اگر Δr را خیلی کوچک در نظر بگیریم، می توان آن استوانه را طوری بُرید تا به شکل یک صفحه نازک مستطیلی در بیاید:



پس ما حجم این استوانه نازک را با این مستطیل تخمین می زنیم. واضح است که هرچه Δr کوچک تر باشد، این تخمین دقیق تر است. در شکل بالا، $b = e^{-r^2}$ و $\ell = 2\pi r$. پس حجم این پوسته استوانه ای برابر است با $2\pi r e^{-r^2} \Delta r$. اگر نمودار را به این استوانه های خیلی نازک تقسیم کنیم و ضخامت هر استوانه را dr در نظر بگیریم، حجم کل زیر نمودار برابر با جمع حجم کل این استوانه ها است:

$$\begin{aligned} \int_0^\infty 2\pi r e^{-r^2} dr &= \int_0^\infty \pi e^{-x} dx \\ &= -\pi [e^{-x}]_0^\infty = 0 - (-\pi e^0) = \pi \\ \Rightarrow \int_{-\infty}^\infty e^{-x^2} dx &= \sqrt{\pi} \end{aligned}$$

دایره به قطرش در توزیع‌های احتمالی حاکم بر طبیعت وجود دارد، نباید خیلی موضوعی عجیب و باورنکردنی باشد.



$$\Rightarrow f(x, 0) = A(x) = f(0, x) = B(x)C(0) = B(0)C(x)$$

$$\Rightarrow \frac{B(x)}{C(x)} = \frac{B(0)}{C(0)} = \text{constant}$$

پس توابع B و C توابع یکسانی هستند و فقط ضریب ثابت متفاوتی پشت‌شان است. ما برای سادگی می‌توانیم فرض کنیم که توابع دقیقاً یکسانی هستند؛ زیرا طبق تعریف‌شان، فقط لازم است که عبارت $f(x, y) = B(x)C(y)$ برقرار باشد، و این معادله می‌تواند طوری تعریف شود که این ضریب پشت B و C یکسان باشد. همچنین در معادلات بالا، ما نشان دادیم که $f(x, 0) = A(x) = B(x)B(0)$ ، پس A و B نیز توابع یکسانی هستند و فقط ضریب پشت‌شان متفاوت است.

حال برای این که محاسبات ما راحت‌تر باشد، فرض می‌کنیم $B(0) = 1$ است. اگر این فرض ما برقرار نباشد، تنها اتفاقی که رخ می‌دهد این است که تابعی که به دست می‌آوریم به یک ضریب ثابت اضافه‌تر در پشتش نیاز دارد و می‌توان این ضریب اضافه‌تر را با نرمالایز کردن تابع به دست آورد. پس می‌توانیم بگوییم که A و B توابع یکسانی هستند و در نتیجه:

$$f(x, y) = A(x)A(y) = A(\sqrt{x^2 + y^2}) = A(r)$$

حال ما تابع $h(x^2) = A(x)$ را تعریف می‌کنیم. واضح است که دامنه h ، اعداد نامنفی است. دقت کنید که خاصیت بالا (که برای A بود)، تبدیل به این خاصیت برای تابع h می‌شود:

$$h(a + b) = h(a)h(b)$$

به راحتی می‌توان نشان داد که این خاصیت h برای هر تعداد عدد برقرار است. یعنی:

$$h(x_1 + x_2 + \dots + x_n) = h(x_1)h(x_2) \dots h(x_n)$$

پس به ازای اعداد طبیعی داریم $h(n) = h(1)^n$. می‌توان نشان داد که به ازای هر عدد گویای نامنفی مثل $\frac{p}{q}$ داریم:

$$h\left(\frac{p}{q}\right) = h\left(\frac{p}{q}\right)^q = h(1)^p \Rightarrow h\left(\frac{p}{q}\right) = h(1)^{\frac{p}{q}}$$

اگر بخواهیم تابع h پیوسته باشد، لازم است که این خاصیت برای اعداد گنگ هم برقرار باشد. پس به صورت کلی‌تر با تعریف $c = h(1)$ داریم:

$$h(x) = c^x = (e^{\ln(c)})^x = e^{x \ln(c)}$$

پس لازم است که $h(x)$ به فرم e^{tx} باشد، پس $A(x)$ به فرم e^{tx^2} است، و در نتیجه داریم:

$$f(x, y) = e^{t(x^2 + y^2)}$$

پس دیدیم که تابع $e^{-(x^2 + y^2)}$ خیلی تابع عجیبی نیست و این خانواده از توابع، تنها توزیع پیوسته‌ای است که دو شرط خیلی معمولی را هم‌زمان دارد. پس حضور این تابع در توزیع‌های احتمالی‌ای که در طبیعت دیده می‌شوند خیلی موضوع عجیبی نیست. همچنین دیدیم که در انتگرال $e^{-(x^2 + y^2)}$ چطور عدد پی پیدا شد. پس این که نسبت محیط

حال توانستیم ببینیم که اصلاً عدد پی از کجا در این انتگرال پیدایش شد. اما راستش را بگوییم، احتمالاً خیلی قانع‌کننده نباشد. اول از همه این که ما در محاسبه انتگرال ظاهراً بدون هیچ دلیلی مسئله را از یک تابع تک متغیره به یک تابع دو متغیره تبدیل کردیم. مشکل دوم هم این است که حتی اگر بپذیریم انتگرال $e^{-(x^2 + y^2)}$ روی $\mathbb{R}^2$ مساوی با عدد پی می‌شود، این که چرا مسئله اولیه را یک بعد بالاتر بردیم تا به این تابع برسیم، به اندازه وجود عدد پی در توزیع نرمال عجیب به نظر می‌رسد.

برای این که این موضوع و همچنین تابع $e^{-(x^2 + y^2)}$ کمی طبیعی‌تر به نظر برسند، نشان می‌دهیم که اگر روی کل توزیع‌های احتمالی دو متغیره و پیوسته، دو شرط خیلی ساده و طبیعی بگذاریم، به همین تابع می‌رسیم، و نه هیچ تابع دیگری.

فرض کنید یک تابع چگالی احتمال را می‌خواهید پیدا کنید که این دو خاصیت را داشته باشد:

۱- این تابع فقط به فاصله از مبدأ وابسته باشد؛ به زبان هندسی یعنی می‌توانید محور مختصات را حول مبدأ دوران بدهید، بدون این که توزیع تغییری کند. به زبان ریاضی:

۲- مقدار x و y در این توزیع از هم مستقل باشند. یعنی اگر یک نمونه از این توزیع بگیریم، دانستن مقدار x این نقطه هیچ اطلاعاتی از مقدار y این نقطه به ما ندهد. به زبان ریاضی:

همچنین فرض می‌کنیم $f(x, y)$ پیوسته است، چون تابعی که در طبیعت پیدا می‌شوند، اکثراً پیوسته هستند (یادآوری می‌کنیم که هدف ما پیدا کردن ارتباط $e^{-(x^2 + y^2)}$ و به همراهش عدد پی با توزیع‌های آماری حاکم بر طبیعت است).

$$f(x, y) = B(x)C(y)$$

همچنین فرض می‌کنیم $f(x, y)$ پیوسته است، چون تابعی که در طبیعت پیدا می‌شوند، اکثراً پیوسته هستند (یادآوری می‌کنیم که هدف ما پیدا کردن ارتباط $e^{-(x^2 + y^2)}$ و به همراهش عدد پی با توزیع‌های آماری حاکم بر طبیعت است).

حال نشان می‌دهیم $e^{-(x^2 + y^2)}$ تنها تابع پیوسته‌ای است که آن دو شرط بالا را دارد.

$$\text{از شرط دوم می‌دانیم که } f(x, y) = B(x)C(y)$$

حال این توابع را به ازای ورودی‌های متفاوت بررسی می‌کنیم:

$$f(0, y) = B(0)C(y) = A(|y|) = A(y)$$

$$f(x, 0) = B(x)C(0) = A(|x|) = A(x)$$

کلید افزایش کارایی در پردازنده‌ها: بافر مقصد انشعاب یا BTB

برنامه‌های کاربردی دنیای واقعی دربرگیرنده برنامه‌های مرکز داده‌ها^۱ شامل حجم عظیمی از دستورات^۳ هستند. این حجم زیاد از دستورات، شامل تعداد قابل توجهی دستورات انشعاب^۴ هستند که موجب می‌شود عدم برخورد^۵ در حافظه نهان دستورات^۶ بالا برود.

یادآوری!

حافظه نهان یا Cache چیست؟

حافظه نهان یک حافظه موقتی است که داده‌های پرکاربرد که در یک بازه زمانی (به این نزدیکی زمانی دسترسی‌ها، محلیت زمانی یا Temporal locality می‌گویند) یا در یک بازه مکانی (از حیث قرارگیری در حافظه، به این نزدیکی، Spatial locality می‌گویند) را در خود نگه می‌دارد تا نیاز به دسترسی کندتر و طولانی‌تر به حافظه اصلی (RAM) را کمتر کند و سرعت اجرای دستورات بالاتر برود.

برای غلبه بر این میزان بالای عدم برخورد در حافظه نهان دستورات، راه‌حل‌های متنوعی ارائه شده‌است که یکی از این راه‌ها FDIP^۷ است. این واحد در پردازنده‌ها، با پیش‌واکشی^۸ دستورهای آینده، آن‌ها را به حافظه نهان دستورات برده تا در زمان اجرای واقعی آماده باشند. نکته حائز اهمیت در پیش‌واکشی دستورات، توجه به دستورات انشعاب شرطی^۹ و غیرشرطی^{۱۰} است. برای دستورات شرطی، طبعاً بخشی تحت عنوان پیش‌بینی‌کننده انشعاب^{۱۱} نیاز داریم که بتواند رخ دادن^{۱۲} یا ندادن^{۱۳} دستور انشعاب شرطی را پیش‌بینی کند. این پیش‌بینی درست در پیش‌واکشی کمک کند، به گونه‌ای که پیش‌بینی اشتباه می‌تواند مجموعه دستورات اشتباهی را وارد حافظه نهان شود. همچنین در هر دو نوع دستورات انشعاب شرطی و غیرشرطی، نیاز است مقصد انشعاب برای پیش‌واکشی مشخص باشد تا بدانیم که پس از این دستور پرش، به چه آدرسی می‌رویم. به واحدی که

مقصد (احتمالی) دستورات انشعاب را در خود نگهداری می‌کند، بافر مقصد انشعاب می‌گویند.

همان‌طور که مشخص است، توانایی پیش‌واکشی FDIP کاملاً وابسته به دقت BTB است. از آن‌جا که ظرفیت BTB محدود است و نمی‌توان مقصد تمام دستورات انشعاب را در آن ذخیره کرد، شاهد عدم برخورد در BTB خواهیم بود.

افزودن حجم BTB یک پردازنده، در هنگام طراحی پردازنده‌ها دچار محدودیت است، زیرا میزان حافظه ایستا^{۱۴} روی تراشه محدود است؛ که این حافظه به عنوان حافظه نهان دستورات و داده‌ها و همچنین BTB استفاده می‌شود. در زمانی که این متن نوشته می‌شود، پردازنده با بیشترین اندازه BTB در بازار، IBM z15 است که دارای یک BTB دولایه، شامل 16K مدخل (BTB Entry) برای لایه اول^{۱۵} و 128K مدخل برای لایه دوم^{۱۶} است. شاید در نگاه اول 144K مدخل برای BTB پردازنده کافی به نظر برسد که شاید این‌طور هم هست! اما باید توجه شود که قیمت این بزرگ‌رایانه^{۱۷} شرکت IBM هم بسیار چشم‌گیر است! در سیستم‌های رایج و ارزان‌تر اندازه BTB در حدود زیر ۱۰k مدخل است که شاید برای برنامه‌های چند مگابایتی (یا گیگابایتی) حال حاضر، کافی نباشد. پیش از بیان روش‌هایی برای عملکرد بهتر BTB و برخورد^{۱۸} بیشتر، به ساختار کلی BTB و FDIP اشاره می‌کنم.

همان‌طور که در تصویر ۱ مشخص است، تاریخچه دستورات^{۱۹} و PC^{۲۰} وارد واحد پیش‌بینی انشعاب^{۲۱} می‌شود که BTB در این واحد قرار دارد. برای دستوراتی نظیر call و return ها از پشته آدرس بازگشت^{۲۲} استفاده می‌شود و همچنین برای پرش‌هایی که مقصد آن‌ها از ثبت‌ها^{۲۳} یا حافظه خوانده می‌شود

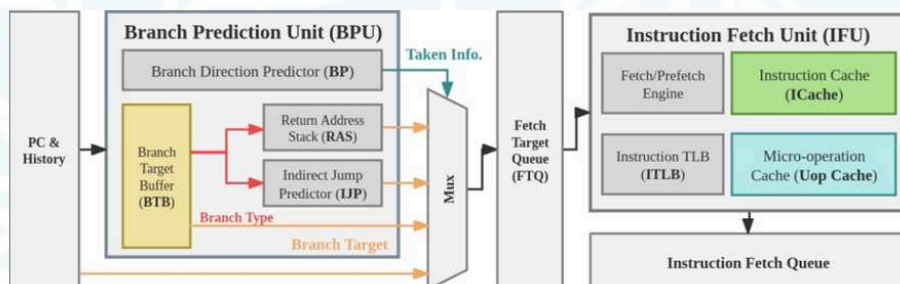
(که اصطلاحاً به آن‌ها پرش غیرمستقیم می‌گوییم)، از JIP^{۲۴} استفاده می‌شود که خود این بخش، برای پیش‌بینی مقصد از حافظه نهان مقصد^{۲۵} یا تاریخچه مسیر^{۲۶} بهره می‌برد. این اطلاعات به همراه اطلاعات BTB و نتیجه پیش‌بینی دستور انشعاب به یک مالتی‌پلکسر^{۲۷} وارد شده و نتیجه نهایی به عنوان آدرس بعدی برای پیش‌واکشی وارد صف مقصد واکشی^{۲۸} می‌شود. در نهایت این آدرس به واحد واکشی دستور^{۲۹} وارد شده تا واکشی/پیش‌واکشی صورت گیرد.

ساختار کلی BTB: بافر مقصد انشعاب چه چیزهایی را نگه می‌دارد؟

BTB اطلاعاتی درباره دستورات انشعابی که حداقل یک‌بار واکشی شده‌اند نگه می‌دارد که شامل نوع دستور انشعاب (پرش مستقیم^{۳۰}، پرش غیر مستقیم^{۳۱}، return، call و...)، مقصد دستور انشعاب^{۳۲}، بیت‌های برجسب^{۳۳} که نقشی مشابه بیت‌های برجسب در حافظه‌های نهان را ایفا می‌کنند و برخی اطلاعات دیگر مانند سیاست جایگزینی^{۳۴} است. دسترسی به BTB درست مانند حافظه نهان است. بنابراین پیش‌بینی می‌شود که مانند حافظه‌های نهان، از سیاست‌های جایگزینی برای قراردادن دستور انشعاب جدید، در صورت پر بودن جایگاه‌های مناسب برای ذخیره‌سازی آن دستور انشعاب در BTB استفاده کنیم.

در زمینه بهبود عملکرد BTB چه کارهایی انجام شده‌است؟

یکی از مواردی که برای بهبود نرخ برخورد در BTBها انجام شده‌است، ابداع



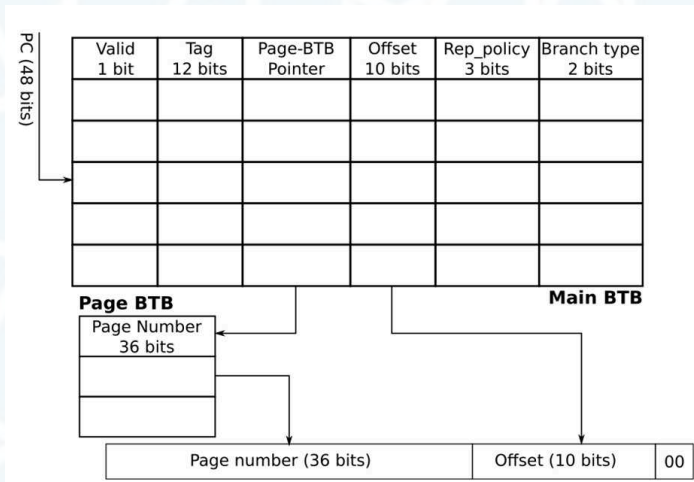
تصویر ۱: ساختار کلی پیش‌واکشی دستورات

Valid: 1	Tag: 12	Type: 2	Target: 46	Rep_policy: 3
----------	---------	---------	------------	---------------

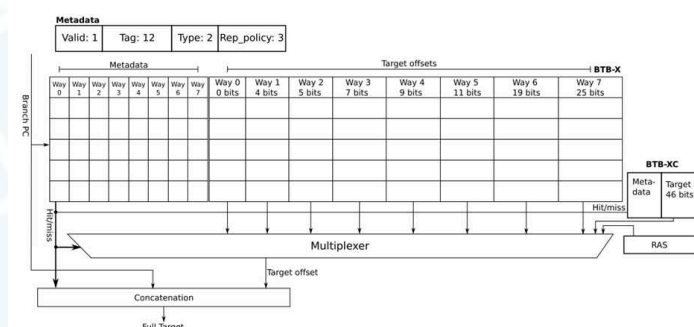
تصویر ۲

Bit position	48	...	9	8	7	6	5	4	3	2	1
Branch PC	0	...	1	0	1	1	0	1	0	0	0
Branch Target	0	...	1	0	1	1	1	1	0	0	0
Target Offset		...					1	1	0	0	0

تصویر ۳



تصویر ۴



تصویر ۵

الگوریتم‌ها^{۳۵} و سیاست‌های جایگزینی است. برخی از این سیاست‌های جایگزینی با استفاده از تاریخچه دستورات گذشته، میزان اهمیت دستورات انشعاب را با روابطی که ارائه می‌دهند، محاسبه کرده و در هنگام جایگزینی، دستورات انشعاب با اهمیت کمتر (آن‌هایی که کمتر دسترسی می‌خورند) را از BTB خارج کرده و دستور جدید را جایگزین می‌کنند. همچنین عده‌ای با پیاده‌سازی الگوریتم بهینه Belady^{۳۶} بر روی تاریخچه دستورات گذشته، قصد دارند عملکردی نیمه‌بهینه^{۳۷} داشته باشند. برای مطالعه بیشتر درباره جزئیات این روش، به مقاله Thermometer^{۳۸} مراجعه کنید.

علاوه بر ارائه سیاست‌های جایگزینی نوین، روش‌های دیگری نظیر تغییر ساختار کلی BTB‌ها می‌تواند مؤثر باشد. برای مثال، یکی از اقدامات انجام‌شده در این زمینه، چندلایه کردن BTB‌هاست. همانند حافظه نهان چندلایه (شامل L1 و L2) یا صفحه‌بندی چندسطحی^{۳۸}، امکان پیاده‌سازی BTB چندلایه نیز وجود دارد. در این روش، یک BTB اولیه^{۳۹} داریم که offset آدرس مقصد دستور انشعاب را مشخص می‌کند و شماره صفحه آدرس مقصد، توسط یک Page BTB تعیین می‌شود. بدین صورت که یک اشاره‌گر از مدخل مربوطه در BTB اولیه به یک سطر در Page BTB وجود دارد که شماره صفحه مربوط به آدرس مقصد را نگهداری می‌کند. حتی امکان پیاده‌سازی این ساختار با بیش از دو لایه نیز وجود دارد. اما از آن‌جا که دسترسی چندلایه احتمالاً زمان بیشتری نسبت به دسترسی به یک لایه صرف می‌کند، این روش باید با در نظر گرفتن سربار^{۴۰} ناشی از افزایش زمان دسترسی و همچنین قابلیت استفاده بهینه از حافظه on-chip SRAM برای BTB پردازنده و ذخیره‌سازی تعداد بیشتری دستور انشعاب، به عنوان یک مبادله^{۴۱} انتخاب شود. (یادآوری: SRAM که عمدتاً برای حافظه نهان سیستم‌ها استفاده می‌شود، از DRAM^{۴۲} مورد استفاده در حافظه اصلی سریع‌تر است، اما هزینه بالاتری نیز دارد.)

روش دیگر برای تغییر ساختار BTB، استفاده از BTB‌های مجموعه-انجمی^{۴۳} است. پیش از پرداختن به این موضوع، به تصویر ۳ توجه کنید.

همان‌طور که مشاهده می‌شود، فرض کردیم که یک سیستم با آدرس‌دهی بایت‌به‌بایت^{۴۴} داریم که فضای آدرس‌دهی آن، شامل آدرس‌های ۴۸ بیتی است. دستورات این سیستم، مضر ۴ بایت^{۴۵} هستند و این یعنی دو بیت سمت راست آدرس آن‌ها صفر است. همان‌طور که در این مثال مشاهده می‌شود، آدرس مقصد دستور انشعاب در بیت‌های

در راه‌های با شماره ۶ و ۷ قرار گیرد. برای اطلاع بیشتر از روش ارائه‌شده در مقاله مربوطه، به مقاله BTB-X^{۴۶} مراجعه کنید!

ادامه راه

شاید به نظر برسد که میزان ایده‌ها و نوآوری‌های انجام‌شده بر روی BTB‌ها به اندازه‌ای بوده که نیاز فعلی پردازنده‌ها و بارکاری‌ها^{۴۷} را برطرف کند، اما با توجه به رشد روزافزون برنامه‌های کاربردی اجراشده بر روی مرکزهای داده، توجه به استفاده از تکنیک‌های جدید برای بهبود عملکرد BTB همچنان ضروری به نظر می‌رسد. بنابراین پژوهشگران همچنان به دنبال راهکارهای جدید برای بهبود عملکرد BTB خواهند بود و این موضوع، می‌تواند زمینه پژوهشی خوبی برای علاقه‌مندان به معماری کامپیوتر باشد!

منابع:

- Thermometer: Profile-Guided BTB Replacement for Data Center Applications
- A Storage-Effective BTB Organization for Servers



پروفسور علی اکبر جلالی



Tag.۳۳
Replacement Policy.۳۴
Algorithms.۳۵
Belady's anomaly algorithm.۳۴
Sub-optimal.۳۵
multi-level paging.۳۶
Main BTB.۳۹
Overhead.۴۰
Trade-Off.۴۱
DRAM (Dynamic Random-Access Memory).۴۲
Set-Associative.۴۳
Byte-Addressable.۴۴
4-Byte-Aligned.۴۵
Way.۴۶
Workloads.۴۷

Mainframe.۱۷
Hit.۱۸
History.۱۹
Program Counter.۲۰
Branch Prediction Unit (BPU).۲۱
Return Address Stack.۲۲
Register.۲۳
Indirect Jump Predictor.۲۴
Target Cache.۲۵
Path History.۲۶
Multiplexer.۲۷
Fetch Target Queue (FTQ).۲۸
Instruction Fetch Unit (IFU).۲۹
Direct jump.۳۰
Indirect jump.۳۱
Branch Target Instruction.۳۲

Branch Target Buffer (BTB).۱
Data center applications.۲
Instructions.۳
Branch instruction.۴
Miss.۵
I-Cache.۶
Fetch Directed Instruction Prefetching.۷
Prefetching.۸
Conditional Branch.۹
Unconditional Branch.۱۰
Branch Predictor.۱۱
Taken.۱۲
Not Taken.۱۳
Static Random-Access Memory.۱۴
L1 (Level 1).۱۵
L2 (Level 2).۱۶

راوی آینده

آذر ۱۳۳۱، کوچه پس‌کوچه‌های روستای شاهکوه واقع در شهر شاهرود، آرام و خاموش، شاهد تولد پسری بودند که سرنوشتش با فردای مدرن گره خورده بود. علی اکبر جلالی کودکی‌اش در همان کوچه‌ها گذشت، اما رویاهایش از دیوارهای گلی فراتر می‌رفت. او با قبولی در تهران، نخستین گام را در مسیری برداشت که به ماجراجویی بزرگش بدل شد.

علی اکبر جلالی پس از ورود به جهاد سازندگی، به روستای پدری‌اش بازگشت و شروع به جمع‌آوری داده‌های دقیق جمعیتی و کشاورزی کرد؛ داده‌هایی که بعدها خودش آن را «اولین قدم‌های آ» در زندگی‌اش می‌نامید. مرکز آماری که طراحی کرد، به‌عنوان دقیق‌ترین شناسنامه اطلاعاتی روستایی شناخته شد.

کمی بعد به پیشنهاد دانشگاه و به دلیل معدل بالایش، راهی آمریکا شد. مقطع ارشد را در دانشگاه اوکلاهاما گذراند؛ جایی که نخستین بار با فناوری کامپیوترهای IBM آشنا شد. در مقطع دکتری و پس‌ادکتری در دانشگاه ویرجینیای غربی^۱، نه تنها به پژوهش ادامه داد، بلکه به نمایندگی از کمپانی متلب^۲ به تدریس در کلاس‌های آمریکایی پرداخت. خودش از آن دوران این‌طور تعریف می‌کند:

«من توی مدرسه حتی نمی‌توانستم کلمه people را تلفظ کنم، حالا در آمریکا به شاگردهای آمریکایی درس می‌دهم. خدایا عظمتت را شکر!»

پس از پایان تحصیل به ایران بازگشت و به‌عنوان استاد دانشگاه علم و صنعت، مأموریت تازه‌ای آغاز کرد. در همان سال‌ها، در حالی که بسیاری حتی نام اینترنت را نشنیده بودند، او آینده‌ای متفاوت می‌دید. طی چندین سخنرانی از «جهان مدرن» گفت و نظریه‌ای را که بعدها «عصر مجازی» یا موج چهارم تغییرات نام گرفت، مطرح کرد؛ نظریه‌ای که حتی با آلوین تافلر^۳، نویسنده کتاب «موج سوم»، در نشستی آن را به اشتراک گذاشت.

در سال ۱۳۷۹، روستای شاهکوه، زادگاهش را به شبکه جهانی متصل کرد و نخستین روستای اینترنتی ایران را بنا نهاد. یک سال بعد، با همکاری دانشگاه علم و صنعت، پروژه شهر الکترونیک را در جزیره کیش آغاز کرد. جالب این‌که، ایده «دبی اینترنت سی»^۴ نیز در همان نمایشگاه شکل گرفت، اما برخلاف دبی، پروژه کیش به دلیل سنگ‌اندازی‌ها و بی‌تدبیری‌های مسئولین بالادستی ناتمام ماند.

او نه تنها در حوزه اینترنت، بلکه در فناوری‌های دیگر نیز پیشگام بود؛ از مشاوره به هلال احمر و ساخت پرینتر سه بعدی برای تولید پای مصنوعی تا طراحی دستگاه همودیالیز بومی. اما حتی در این مسیر هم با مخالفت‌هایی روبه‌رو شد؛ جایی که به او گفتند: «نظم بازرگانی ما را به هم زدید؛ ما این دستگاه را از خارج وارد می‌کردیم.»

با وجود این موانع، جلالی دست از تلاش برنداشت و بارها در سخنرانی‌هایش هشدار داد:

«بیست سال پیش به مسئولان گفتیم اینترنت اقتصاد آینده است، هی فیلترش کردند، هی بستندش، گفتند این بده... امروز همان‌ها گرفتارش شده‌اند و مجبورند تسلیمش شوند.»

او بارها تأکید کرد:

«در جهان آینده، هر کسی که به اطلاعات بیشتری دسترسی داشته باشد، جهان را بهتر کنترل می‌کند.»

و با چشمانی اشک‌آلود، در برابر نقاشی‌ای که پشت سرش بود، آرام گفت:

«گاهی وقتا آدم با کلی زحمت این نقاشیو می‌کشه، به نمایش درمیاره... ولی بهت تهمت می‌زنن که این فساد.»

byte

People who are
really serious about
software should
make their own
hardware.

Alan Kay



At a certain point,
the services that
you build around the
hardware become more
important than the
hardware itself.

Nick Woodman

It's hardware
that makes a
machine fast.
It's software
that makes a
fast machine
slow.

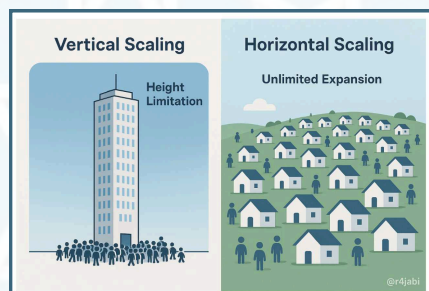
Craig Bruce





دریافت کند و افزودن این منابع معمولاً با هزینه‌هایی تصاعدی همراه است.

در مقابل آن، مقیاس‌پذیری افقی یعنی این که سیستم را بین چند گره پخش کنیم - مثل ساختن یک شهر با هزاران خانه کوچک. بدین شکل که هر زمان جمعیت بیشتر شود، فقط کافی است که خانه‌های بیشتری اضافه کنیم. ساده، قابل توسعه و پایدار.



حالا برگردیم سراغ دیتابیس‌هایی که همه با آن‌ها آشنا هستیم: دیتابیس‌های رابطه‌ای نظیر MySQL و PostgreSQL.

این دیتابیس‌ها دقیقاً مانند آسمان‌خراش هستند. برای تعداد کمی از کاربران مناسب هستند، اما وقتی تعداد کاربران افزایش یابد به مشکل می‌خورند. تقریباً مقیاس‌پذیری افقی ممکن نیست، چون طراحی متمرکزی دارند و معمولاً مدل CA را انتخاب کرده‌اند - یعنی داده دقیق و همیشه در دسترس، ولی فقط وقتی که شبکه بدون مشکل باشد. و این یعنی در سیستم‌های توزیع‌شده، نمی‌توان روی آن‌ها حساب باز کرد.

دیتابیس‌هایی که قوانین را تغییر دادند!

این جاست که پای پایگاه‌های داده‌ای مانند MongoDB و Cassandra به میان می‌آید. آن‌ها گفتند: «قوانین قدیمی کافی نیستند، باید جور دیگری فکر کنیم.»

اما این تغییر رویکرد بدون هزینه نیست. دیگر نمی‌توانید بی‌دردسر و بدون برنامه‌ریزی دقیق، جدول طراحی کنید. باید مانند یک شطرنج‌باز چند حرکت جلوتر را ببینید: کاربران چه سؤالاتی خواهند پرسید؟ داده‌ها چگونه پراکنده می‌شوند؟ وقتی سیستم مقیاس می‌گیرد، چه چالش‌هایی پیش می‌آید؟

۱. یکپارچگی

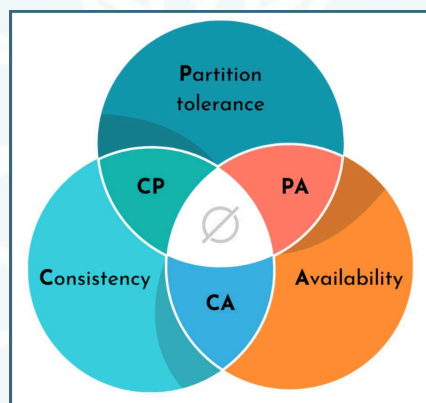
همه گره‌ها در یک لحظه، یک نمای یکسان از داده دارند. یعنی اگر روی گره A چیزی write کنید، فوراً از گره B هم همان مقدار read می‌شود.

۲. دسترس‌پذیری

سیستم همیشه به درخواست‌ها پاسخ می‌دهد (البته نه لزوماً با آخرین داده). حتی اگر بخش‌هایی از شبکه از کار بیفتند.

۳. تحمل پارتیشن

وقتی ارتباط شبکه بین گره‌ها قطع می‌شود، سیستم همچنان کار می‌کند.



اما در دنیای واقعی و سیستم‌های توزیع‌شده، قطع شدن ارتباط بین گره‌ها اجتناب‌ناپذیر است. شبکه حتماً گاهی مشکل پیدا می‌کند، پس عملاً شما باید بین ترکیب‌های CA و PA یکی را انتخاب کنید.

چرا دیتابیس‌های قدیمی کافی نیستند؟

یکی از مهم‌ترین چالش‌ها در طراحی سیستم‌های بزرگ، انتخاب کردن نوع مقیاس‌پذیری است. دو رویکرد اصلی داریم:

۱. مقیاس‌پذیری عمودی

۲. مقیاس‌پذیری افقی

مقیاس‌پذیری عمودی به معنای تقویت یک سیستم واحد از طریق افزودن منابع سخت‌افزاری مانند حافظه و پردازنده است - مانند این که یک آسمان‌خراش بسازیم و هر زمان که نیاز داشتیم، یک طبقه به آن اضافه کنیم. این روش در ابتدا منطقی به نظر می‌رسد، اما با افزایش تعداد کاربران به محدودیت‌هایی برمی‌خوریم. زیرا هر سیستم تنها تا اندازه مشخصی می‌تواند منابع بیشتری

در هر ثانیه، نتفلیکس ۱۵ هزار ساعت ویدیو نمایش می‌دهد. اینستاگرام روزانه ۹۵ میلیون عکس پردازش می‌کند. وقتی روی دکمه جست‌وجوی گوگل کلیک می‌کنید، بیش از هزار کامپیوتر در سراسر جهان با یکدیگر مسابقه می‌دهند تا در کسری از ثانیه پاسخ شما را پیدا کنند.

تا حالا به این فکر کرده‌اید که چطور همه این‌ها، بدون هیچ مشکلی انجام می‌شوند؟ این دقیقاً دنیای سیستم‌های توزیع‌شده است. اگر تا الان فکر می‌کردید پروژه «برنامه‌نویسی پیشرفته» چالش بزرگی بود، خبر بد این است که تازه ماجرا شروع شده است. چون حالا می‌خواهیم درباره سیستم‌هایی صحبت کنیم که روی هزاران دستگاه پراکنده، در قاره‌های مختلف کار می‌کنند، هرگز خاموش نمی‌شوند و باید با خراب شدن قطعات و سیستم‌ها کنار بیایند و همه این‌ها در حالی است که میلیاردها کاربر منتظر پاسخ فوری هستند.

یک مهمانی برای یک میلیون نفر

بهترین راه برای درک این موضوع یک مثال است. تصور کنید می‌خواهید مهمانی برگزار کنید؛ اما نه یک مهمانی معمولی - مهمانی برای یک میلیون نفر که در شهرهای مختلف زندگی می‌کنند. حالا تصور کنید همه باید هم‌زمان درباره یک موضوع صحبت کنند، اما هر لحظه ممکن است بعضی از آن‌ها ناپدید شوند یا اتصال‌شان قطع شود و مهم‌تر از همه، این مهمانی هرگز تمام نمی‌شود.

خیلی پیچیده به نظر می‌رسد، نه؟ اما این دقیقاً همان کاری است که سیستم‌های توزیع‌شده هر روز انجام می‌دهند و جالب این جاست که نه تنها این کار ممکن است، بلکه به قدری روان اتفاق می‌افتد که ما حتی متوجه آن نمی‌شویم.

انتخاب‌های سخت

اما این ماجرا یک قانون طلایی دارد که همه چیز را تغییر می‌دهد: CAP Theorem. Eric Brewer در سال ۲۰۰۰ ثابت کرد که در هر سیستم توزیع‌شده، شما فقط می‌توانید دو مورد از سه ویژگی زیر را انتخاب کنید:



سیستم‌های توزیع‌شده یکی از چالش‌برانگیزترین حوزه‌های علوم کامپیوتر هستند، اما همین پیچیدگی است که آن‌ها را جذاب می‌کند. هر مسئله‌ای که حل می‌کنید و هر تصمیمی که می‌گیرید، تأثیر مستقیمی روی عملکرد و تجربه کاربران واقعی دارد.

از طراحی دیتابیس‌های غول‌پیکر گرفته تا مدیریت ترافیک میلیاردری، این حوزه مملو از مسائل جالب و حل‌نشده است. اگر به دنبال چالش‌های فنی واقعی هستید، مطالعه و کار روی سیستم‌های توزیع‌شده می‌تواند آغاز یک مسیر هیجان‌انگیز باشد.

برای شروع این مسیر، منابع زیر را پیشنهاد می‌کنم که می‌توانند دید عمیق‌تری از مفاهیم، معماری‌ها و راه‌حل‌های دنیای واقعی به شما بدهند.

▪ کتاب *Designing Data-Intensive Applications* نوشته Martin Kleppmann

یکی از بهترین منابع برای درک معماری سیستم‌های مدرن، CAP، دیتابیس‌ها و مفاهیم کلیدی این حوزه.

▪ مستندات *The Cassandra Documentation*

مستندات رسمی Cassandra که شامل آموزش‌ها، مفاهیم کلیدی و راهنمای نصب و استفاده است.

▪ وب‌سایت *MongoDB University*

دوره‌های رایگان آموزشی از سوی تیم MongoDB برای آشنایی با مفاهیم پایگاه‌های داده غیررابطه‌ای.

▪ مجموعه سخنرانی‌های *Distributed Systems* از دانشگاه MIT

اگر به دنبال منابع آکادمیک هستید، این دوره یکی از بهترین‌هاست.

امیدوارم این منابع نقطه شروع خوبی برای ماجراجویی شما در دنیای سیستم‌های توزیع‌شده باشند.

دانستنی

MongoDB یک پایگاه‌داده مستندمحور است؛ به جای جدول‌های سنتی، داده‌ها در قالب اسناد JSON ماندنی ذخیره می‌شوند. این ساختار، انعطاف زیادی در ذخیره‌سازی انواع داده‌ها فراهم می‌کند و برای سامانه‌هایی با داده‌های متنوع و پویا بسیار مناسب است.

در مقابل، Cassandra برای مدیریت حجم عظیمی از داده‌های توزیع‌شده طراحی شده‌است. این سیستم ابتدا در شرکت فیسبوک توسعه یافت و سپس به Apache واگذار شد. Cassandra از مدل ستونی گسترده استفاده می‌کند و به گونه‌ای طراحی شده که بتواند در مقیاس‌های بالا، عملکردی پایدار و سریع داشته باشد.

برای مثال، در Cassandra مفهومی به نام Join وجود ندارد؛ ابتدا ممکن است عجیب به نظر برسد، اما کاملاً منطقی است؛ وقتی داده‌ها روی ده‌ها یا صدها سرور پراکنده شده‌اند، انجام عملیات Join بسیار پیچیده و کند خواهد بود.

پذیرش زیبایی نبود قطعت

فرض کنید از Cassandra استفاده می‌کنید. برخلاف پایگاه‌داده‌های سنتی، Cassandra معماری‌ای مبتنی بر گره‌های مستقل دارد؛ یعنی داده‌ها میان چندین گره که ممکن است در نقاط مختلف دنیا قرار داشته باشند، توزیع می‌شوند. هر گره بخشی از داده‌ها را نگه می‌دارد و به صورت مستقل می‌تواند پاسخ‌گو باشد.

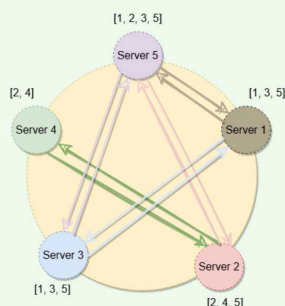
Cassandra مدل PA را در مثلث CAP انتخاب کرده است - یعنی شما می‌پذیرید که گاهی اطلاعات کمی قدیمی باشند، اما در عوض سیستم همیشه در دسترس خواهد بود و هیچ‌گاه از کار نمی‌افتد.

مثلاً تصور کنید ویدیویی را در یوتیوب لایک می‌کنید؛ شمارنده فوراً یک واحد افزایش می‌یابد، اما اگر صفحه را بلافاصله رفرش کنید، ممکن است عدد کمی متفاوت باشد، چون همه گره‌ها هنوز به‌روزرسانی نشده‌اند. چند ثانیه بعد، همه به یک عدد مشترک می‌رسند - ولی در این میان، یوتیوب هرگز نمی‌گوید «منتظر بمانید، مشغول هماهنگ‌سازی لایک‌ها هستیم.»

این انتخاب زیبایی خاصی دارد. شما در حال طراحی سیستمی هستید که با واقعیت دنیای فیزیکی کنار آمده - جایی که زمان می‌برد تا اطلاعات منتقل شوند، سرورها گاهی خراب می‌شوند و حالت ایده‌آل وجود ندارد.

در یک سیستم توزیع‌شده مثل Cassandra یا Consul، سرورها باید از وضعیت یکدیگر باخبر باشند؛ کدام گره فعال است؟ کدام داده روی کدام سرور قرار دارد؟ برای این کار از Gossip Protocol استفاده می‌شود. ایده‌اش ساده و الهام‌گرفته از شایعه‌پراکنی است: هر گره به‌طور دوره‌ای با چند گره تصادفی تماس می‌گیرد و وضعیت خودش (مانند بار، داده‌های موجود یا خطاها) را منتقل می‌کند. آن گره‌ها هم همین اطلاعات را به بقیه منتقل می‌کنند. بعد از چند دور و بدون نیاز به یک نود مرکزی، همه سرورها تصویر نسبتاً دقیقی از کل سیستم پیدا می‌کنند. مزیت بزرگ این روش مقیاس‌پذیری و تحمل خطاست؛ چون هیچ نقطه واحدی برای شکست وجود ندارد و شبکه به‌طور طبیعی با خرابی کنار می‌آید. علاوه بر این، Gossip معمولاً از الگوریتم‌های تصادفی و دوره‌ای استفاده می‌کند تا سربار شبکه کمینه شود و در نهایت داده‌های وضعیت با تأخیر کوتاه اما به‌صورت پایدار و مقاوم در کل خوشه منتشر شوند و اگر نودی متقلبان رفتار کند؛ مثلاً از پروتکل پیروی نکند و سعی کند به سایر نودها یا کاربران آسیب برساند این روش باید آن را پیش‌بینی و عملکرد صحیح و کارآمد را تضمین کند.

Every second each server exchanges information with one randomly selected server





حدود یک سال می‌شود که با زبان zig آشنا شده‌ام، زبانی که ادعا می‌کند که «یک C بهتر» است و از وقتی با آن آشنا شده‌ام، دیدم را نسبت به خیلی از ابعاد برنامه‌نویسی گسترش داده‌است. این زبان ابعاد بسیار گسترده‌ای دارد، اما برای این متن تصمیم گرفتم که با تمرکز بر «مدیریت حافظه»، «Comptime» و «System build» در zig برای‌تان بنویسم.

مدیریت حافظه

یکی از جنبه‌های مهم هر زبان برنامه‌نویسی، نحوه برخورد با حافظه است؛ چرا که در رایانه‌های امروزی، چه در پردازنده‌های گرافیکی و چه در پردازنده‌های مرکزی، با مسئله حافظه دست‌وپنجه نرم می‌کنیم و گلوگاه اصلی هر برنامه یا پردازنده، سرعت دسترسی به حافظه است.

یکی از ایده‌های اساسی این زبان این است که هیچ «تخصیص‌دهنده حافظه» مرکزی وجود ندارد؛ در عوض، این مفهوم به‌عنوان یک درگاه^۱ مشتق‌شده از توابع alloc و free تعریف می‌شود که پیاده‌سازی‌های متفاوتی در کتابخانه استاندارد دارد. چند پیاده‌سازی این درگاه را در زیر مشاهده می‌کنید:

std.heap.PageAllocator

تخصیص‌دهنده‌ای که به ازای هر درخواست، به‌طور مستقیم فراخوانی‌های سیستمی مرتبط با محیط اجرای برنامه را صدا می‌زند. به‌طور مثال در محیط posix از mmap و munmap استفاده می‌کند.

std.heap.SmpAllocator

تخصیص‌دهنده‌ای که از نظر فلسفه و عملکرد مشابه malloc در زبان C عمل می‌کند.

std.heap.DebugAllocator

تخصیص‌دهنده‌ای که به برنامه‌نویس در دیباگ کردن مشکلات حافظه مانند «آزاد کردن مجدد» و «نشت منابع» کمک می‌کند و به‌طور دقیق محل وقوع این مشکلات را گزارش می‌کند.

std.heap.ArenaAllocator

تخصیص‌دهنده‌ای که مانند یک پشته عمل می‌کند. با استفاده از این تخصیص‌دهنده می‌توانید به سادگی حافظه را مدیریت کنید؛ به این صورت که به جای این‌که نگران طول عمر تک‌تک تخصیص‌ها باشید، می‌توانید به‌طور یک‌جا برای یک Arena طول عمر تعیین کنید. به این صورت، هم طراحی برنامه

صحیح راحت‌تر می‌شود و هم در بسیاری از موارد شاهد تسریع عملکرد خواهیم بود؛ چرا که تخصیص‌دهنده اصلی برنامه نیازی نیست به هزاران یا میلیون‌ها تخصیص خرد فکر کند و آن‌ها را نگه دارد. در عوض کافی است تعدادی محدود از Arena‌ها را نگه دارد.

std.heap.FixedBufferAllocator

تخصیص‌دهنده‌ای که قطعاتی از یک حافظه ثابت را به دیگران تخصیص می‌دهد. از این تخصیص‌دهنده می‌توانید استفاده کنید تا هر تخصیص حافظه دلخواهی را به پشته برنامه منتقل کنید یا این‌که در ابتدای برنامه از روی heap یک قطعه حافظه بگیرید و در ادامه از آن استفاده کنید و در میانه برنامه هیچ تخصیص حافظه‌ای نداشته باشید. به‌طور مثال، TigerBeetle (سریع‌ترین و پایدارترین پایگاه داده تراکنشی) تمامی تخصیص‌های حافظه را در ابتدای برنامه انجام می‌دهد.

به این شکل، کتابخانه‌ها مستقل از یک تخصیص‌دهنده، ثابت خواهند شد و مسئولیت این کار بر عهده استفاده‌کننده قرار می‌گیرد. برای مثال، شما می‌توانید از همه کتابخانه‌ها، حتی بدون وجود heap استفاده کنید. این معماری می‌تواند به بهبود کارایی و افزایش قابلیت انتقال برنامه‌های شما کمک شایانی کند. به‌طور مثال، شما می‌توانید از هر کتابخانه‌ای روی سامانه‌های نهفته^۲ استفاده کنید.

ورودی/خروجی نیز در نسخه ۰.۱۵ و ۰.۱۶ مانند Allocator خواهد شد و در حال حاضر، اجرای کتابخانه‌هایی که از ورودی و خروجی استفاده می‌کنند روی این سامانه‌ها ممکن نیست اما با کامل شدن درگاه ورودی/خروجی این امر میسر خواهد شد.

Comptime

یکی از وجوه تمایز zig نسبت به بسیاری از زبان‌های دیگر این مورد است که می‌توان تقریباً هر کدی را که با ورودی/خروجی و عملیات‌های وابسته به یک بستر خاص (مانند اسمبلی) کاری نداشته باشد، در زمان کامپایل اجرا کرد. علاوه بر این، zig به شما امکان انجام «انعکاس»^۳ روی انواع داده را می‌دهد و به این صورت می‌توانید ویژگی‌های مختلف مانند «فیلدها»^۴ و «متدها»^۵ را بررسی کنید، مقادیر آن‌ها را تغییر دهید یا بخوانید. این

قابلیت، اجرای الگوهای «طراحی داده‌محور» را بسیار ساده می‌کند، بدون آن‌که نگران صحت عملکرد برنامه باشید؛ چرا که تبدیل کد به حالت داده‌محور آن، در زمان کامپایل و توسط الگوریتم شما انجام می‌شود. در ادامه، چند مثال از این ابزار قدرتمند zig را مشاهده می‌کنید:

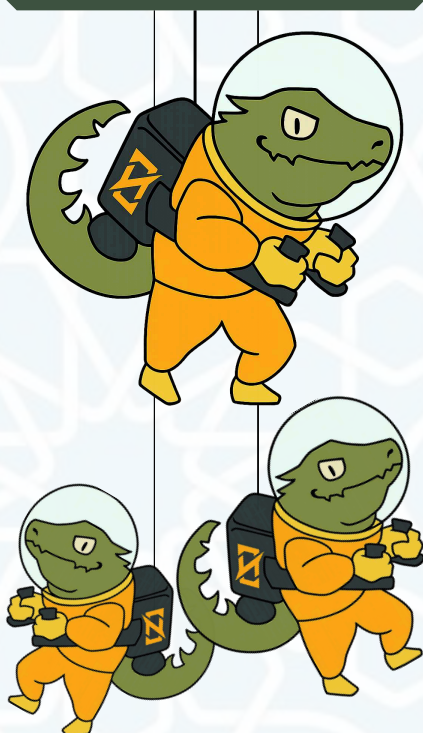
اجرای تابع فیبوناچی در زمان کامپایل

```
const std = @import("std");
pub fn main() !void {
    const fib10 = comptime blk: {
        var fib = [_]usize{ 1, 1 };
        for (0..5) |_| {
            const tmp_fib = fib;
            fib =.{ tmp_fib[1], tmp_fib[0] + tmp_fib[1] };
        }
        break :blk fib[0];
    };
    std.debug.print("{} ", .{fib10});
}
```

داده‌ساختارهای کلی‌سازی‌شده

```
const std = @import("std");
pub fn main() !void {
    const T = GenericType(i32, usize);
    try T.printTypes();
}

fn GenericType(T0: type, T1: type) type {
    return struct {
        a: T0,
        b: T1,
        pub fn printTypes() !void {
            std.debug.print("{} and {} \n", .{ @typeName(T0), @typeName(T1) });
        }
    };
}
```



و با استفاده از کد `build.zig` زیر یک باینری به نام `exe` با لینک کردن تمام فایل‌های `.cpp` درون پوشه `src` به عنوان فایل‌های به زبان `.cpp` توصیف می‌شود:

```
const std = @import("std");
const util = @import("build-util");

pub fn build(b: *std.Build) !void {
    const targ = b.standardTargetOptions(.{});
    const optm = b.standardOptimizeOption(.{});
    const exe = b.addExecutable(.{
        .name = "exe",
        .root_module = b.createModule(.{
            .target = targ,
            .optimize = optm,
            .link_libcpp = true,
        }),
    });
    const srcdir = try b.build_root.handle.openDir("src/",
        .{ .iterate = true });

    exe.root_module.addSourceFiles(.{
        .language = .cpp,
        .root = b.path("src"),
        .files = try util.listFilesRecursive(b.allocator, 16,
            srcdir, ".cpp"),
    });
    exe.root_module.addIncludePath(b.path("src/"));
    b.installArtifact(exe);
}
```

C++ را با یکپارچه‌سازی فرآیندهای `Caching` و مدیریت پکیج‌ها در دو سطح سیستم و کد منبع حل کرده‌است. در نتیجه، با استفاده از `zig` به راحتی می‌توانید پروژه خود را از هر سیستم‌عاملی به هر سیستم هدفی کامپایل کنید و این قابلیت، چالش‌های `Cross Compile` در C و C++ را تا حد بسیار زیادی مرتفع ساخته است.

ساختار سیستم ساخت `zig` به این صورت است که کافی است یک فایل `build.zig` برای توصیف فرایند ساخت، دستورات و گزینه‌های موجود شامل یک تابع `build` داشته باشید. همچنین، برای مشخص کردن نام، برخی ویژگی‌های پروژه و وابستگی‌های آن، یک فایل `build.zig.zon` نیاز است. به علاوه، `zig` آماده‌سازی اولیه این محیط را برای شما با استفاده از دستور

```
zig init
zig fetch --save {PKG-PATH}
zig fetch --save="build-util"
git+https://github.com/MahdiGMK/build-util.zig.git
```

```
const std = @import("std");
pub fn main() !void {
    const x = Cat{};
    const y = Dog{};
    const a0 = Animal.init(x);
    const a1 = Animal.init(y);
    a0.sound();
    a1.sound();
}

const Cat = struct {
    pub fn sound(_: Cat) void {
        std.debug.print("meow!!\n", .{});
    }
};

const Dog = struct {
    pub fn sound(_: Dog) void {
        std.debug.print("woof!!\n", .{});
    }
};

const Animal = union(enum) {
    cat: Cat,
    dog: Dog,
    pub fn init(self: anytype) Animal {
        const options = std.meta.fields(Animal);
        inline fn (options) |fld| {
            if (@typeOf(self) == fld.type)
                return @unionInit(Animal, fld.name, self);
        }
        @compileError("In compatible type of self");
    }
    pub fn sound(self: Animal) void {
        switch (self) {
            inline else => |x| {
                x.sound();
            }
        }
    }
};
```

برای مشاهده مثال‌های بیشتر، می‌توانید به کتابخانه استاندارد `zig` مراجعه کنید. توابعی مانند `std.fmt.print` به طور کامل درون خود زبان پیاده‌سازی شده‌اند و نیازی به از پیش تعیین شدن‌شان درون کامپایلر نیست. همچنین، برای مشاهده یک نمونه داده‌محور، می‌توانید به `std.MultiArrayList` مراجعه کنید و همین‌طور مقالات و ارائه‌های «اندرو کلی»^۴ -سازنده `zig` را مطالعه نمایید.

نگهداری از پروژه‌ها با استفاده از `zig`

همان‌طور که گفته شد، `zig` قصد دارد جایگزین C شود. بدیهی است که نمی‌توان میلیون‌ها و شاید میلیارد‌ها خط کد C و C++ موجود در دنیا را نادیده گرفت. از این رو، `zig`، خود یک کامپایلر C و C++ است و از نظر سهولت استفاده، بسیار راحت‌تر از `gcc` و `clang` عمل می‌کند و پیش‌فرض‌های معقول‌تر و بهینه‌تری نسبت به آن‌ها دارد.

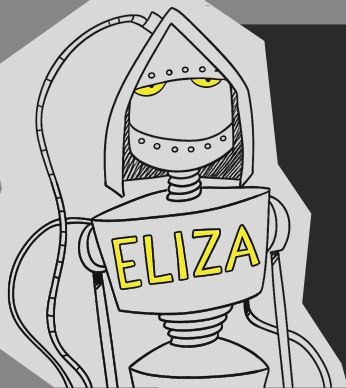
روشی که `zig` برای دستیابی به این هدف در پیش گرفته، این است که علاوه بر بهره‌گیری از LLVM (که پشتوانه اکثر کامپایلرهای نوین است)، کل کتابخانه `libc++` را نیز درون خود دارد و در عمل می‌تواند تابع `main` خود کامپایلر `Clang` را فراخوانی کند! علاوه بر این، `zig` مشکل دیرینه سیستم‌های ساخت^۵ در C و

زبان `zig`، یک زبان برنامه‌نویسی متن‌باز^۴ است که طبق ادعای سایت رسمی آن، با اهدافی همچون قدرت بالا، بهینه بودن و همچنین وضوح سورس‌کد به صورت متن‌باز توسعه یافته‌است. `zig` توسط اندرو کلی در سال ۲۰۱۵ شروع شد و اکنون به نظر می‌رسد که به حجم بحرانی^۵ لازم رسیده‌است. جاه‌طلبی `zig` در تاریخ نرم‌افزار بسیار مهم است: تبدیل شدن به وارث سلطنت طولانی مدت C به عنوان زبانی سطح پایین، قابل حمل و به عنوان استاندارد یک زبان‌های دیگر با آن سنجیده می‌شوند.

ZIG



The Loebner Prize



پیش‌تر با آزمون تورینگ آشنا شدیم مسابقه‌ای که در آن ماشین‌ها محک می‌خورند تا رفتار انسان را وانمود کنند. اما باید بدانیم بازیکنان کمی توانستند موفقیتی کسب کنند و آن‌هایی هم که موفق شدند، بیش‌تر روی پیدا کردن راه‌هایی هوشمندانه برای فریب داور متمرکز شدند تا استفاده از توانایی‌های کامپیوتری. اولین برنامه‌ای که ادعای موفقیت داشت «الایزا» بود که با تقلید از یک روان‌شناس، داوران زیادی را فریب داد؛ این برنامه با پرسیدن سؤال‌های پی‌درپی و تشویق افراد به ادامه گفتگو توانست مکالمه‌ای ظاهراً طبیعی ایجاد کند.

بعدها با برگزاری مسابقات سالانه‌ای مانند «جایزه لوبنر»، تست تورینگ به شکل عملی برگزار شد و حالت جدی‌تری به خود گرفت. یکی از نمونه‌های موفق در این رقابت، برنامه‌ای به نام «کاترین» بود که توانست نظر داوران را جلب کند. این برنامه به شکل چشم‌گیری قادر بود گفت‌وگویی هوشمندانه را ادامه دهد، اما در صورتی که درباره موضوعی خاص به‌ویژه درباره بیل کلینتون صحبت کند؛ به همین دلیل برنامه تلاش می‌کرد مکالمه را به همان سو هدایت کند.



برنده دیگری به نام «یوجین گوستمن» شخصیت یک پسر ۱۳ ساله اوکراینی را شبیه‌سازی می‌کرد. همین طراحی باعث می‌شد تا خطاهای زبانی، گرامر نامعمول و پاسخ‌های غیرمنطقی او طبیعی‌تر به نظر برسند. داوران معمولاً این اشکالات را به تفاوت‌های زبانی و فرهنگی نسبت می‌دادند و همین موضوع موجب شد این برنامه عملکرد قابل‌توجهی در آزمون داشته باشد. با این حال، جایزه لوبنر بیش‌تر بر پایه شبیه‌سازی باورپذیر و فریب هوشمندانه بود تا هوش واقعی و به نظر می‌رسد با وجود پیشرفت‌های امروز در مکالمات هوش مصنوعی، دوباره باید به این پرسش بزرگ بازگردیم: هوشیاری واقعاً چیست؟

انتقادات و پیشنهاد‌های خود را
با ما در میان بگذارید

