الگوریتم‌های آنلاین

کاوه حسینی

بخش اول

مقدمه

در طول ۲۰ سال گذشته الگوریتم‌های آنلاین بسیار مورد توجه واقع بوده‌اند. مسایل آنلاین در بسیاری از حوزه‌های کاربردی مطالعه شده‌اند، از جمله مدیریت منابع در سیستم‌های عامل، داده ساختارها، برنامه‌ریزی، شبکه و امور مالی محاسباتی.

به طور رسمی یک الگوریتم آنلاین دنباله‌ای از درخواست‌های $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ را دریافت می‌کند. این درخواست‌ها باید به ترتیب ورود پاسخ داده شوند. هنگام پاسخ به درخواست $\sigma(t)$ الگوریتم از درخواست‌های $t' > t$، $\sigma(t')$ اطلاع ندارد. پاسخ دادن به هر درخواست هزینه‌ای را می‌طلبد. هدف کمینه کردن هزینه کلی پرداخت شده برای دنباله‌ی درخواست‌هاست. این فرایند را می‌توان به عنوان یک بازی پاسخ به درخواست[1] در نظر گرفت. دشمن درخواست‌ها را تولید می‌کند و الگوریتم بایستی به هر کدام پاسخ دهد. کارایی الگوریتم‌های آنلاین را معمولا به روش تحلیل رقابتی می‌سنجد[۴]. در این روش الگوریتم آنلاین ALG با الگوریتم آفلاین[2] بهینه‌ی OPT که از دنباله‌ی درخواست‌ها، σ، از همان اول اطلاع دارد و می‌تواند در هر مرحله پاسخی با هزینه‌ی کمینه بدهد، مقایسه می‌شود.

تعریف ۱. *فرض کنید σ داده شده و $ALG(\sigma)$ و $(OPT)(\sigma)$(به ترتیب هزینه‌ی الگوریتم‌های ALG و OPT را نشان می‌دهد. الگوریتم ALG را c-رقابتی[3] می‌نامیم اگر ثابت b وجود داشته باشد به طوری که $ALG(\sigma) \leq c.OPT(\sigma) + b$، برای همه‌ی دنباله‌های σ.*

گفتنی است تحلیل رقابتی ابزاری قوی برای تحلیل الگوریتم در بدترین حالت است.

نتایج اولیه

مسئله‌ی صفحه‌بندی[4] یکی از مسایل مهم و احتمالا قدیمی‌ترین مسئله‌ای است که در حوزه‌ی محاسبات تعاملی مطرح شده است. مسئله از تعامل داده توسط CPU با سلسله مراتب حافظه ناشی می‌شود. در این مسئله دارای دو لایه حافظه هستیم. یک حافظه‌ی کم ظرفیت و سریع M_1 و یک با ظرفیت بالاتر ولی سرعت کم M_2. اطلاعات به بخش‌های[5] مساوی تقسیم شده است. CPU به طور مستقیم تنها می‌تواند با حافظه‌ی M_1 تبادل اطلاعات کند. سیستم دنباله‌ای درخواست دریافت می‌کند که هر درخواست به یک بخش از اطلاعات مربوط می‌شود. درخواست را بلافاصله می‌توان جواب داد اگر وتنها اگر بخش مربوطه در M_1 وجود داشته

[1] Request answer game
[2] Offline algorithm
[3] c-Competitive
[4] Paging
[5] Page

باشد، در غیر این صورت یک خطا[۶] رخ می‌دهد. سپس بخش مربوط از حافظه‌ی $M_۲$ به $M_۱$ آورده می‌شود و درخواست پاسخ داده می‌شود. با هر بار کپی کردن یک بخش به $M_۱$ یکی از بخش‌های فعلی $M_۱$ حذف می‌شود تا جا برای بخش جدید باز شود. الگوریتم صفحه‌بندی تعیین می‌کند کدام یک از بخش‌ها حذف شود. این تصمیم هم باید به شکل آنلاین صورت گیرد. هزینه‌ای که تمایل به کمینه کردن آن داریم تعداد خطاهاست.

عمده‌ترین الگوریتم‌های صفحه‌بندی در زیر آورده شده‌اند:

- LRU (اخیرا کمترین استفاده شده[۷]): بخشی را حذف کن که اخیرا کمتر از بقیه استفاده شده است.
- FIFO (اولین ورودی-اولین خروجی[۸]): بخشی را حذف کن که زمان بیشتری نسبت به بقیه در $M_۱$ باقی‌ماند است.
- LIFO (آخرین ورودی-اولین خروجی[۹]): آخرین بخشی را حذف کن که به حافظه وارد شده است.
- LFU (اخیرا کمترین استفاده شده[۱۰]): بخشی را حذف کن که اخیرا از بقیه کمتر استفاده شده است.

سلیتورو تارجان[۴] کارایی دو الگوریتم اول را بررسی کرده‌اند. فرض کنید ظرفیت $M_۱$، k بخش باشد.

قضیه ۲. *LRU و FIFO هردو k-رقابتی هستند.*

اثبات. نشان می‌دهیم LRU، k-رقابتی است. برای الگوریتم FIFO هم به شکل مشابه اثبات می‌شود.

فرض کنید پیکربندی[۱۱] اولیه‌ی الگوریتم MIN (الگوریتم بهینه‌ی آفلاین) و LRU در حافظه‌ی $M_۱$ یکی باشد. بایستی نشان دهیم برای هر k بخش اولیه در حافظه‌ی $M_۱$ و برای هر دنباله‌ی σ، $\sigma C_{\text{LRU}}(\sigma) \leq k.C_{\text{MIN}}(\sigma)$.

عملیات LRU را برای هر دنباله‌ی σ ای خاص بررسی می‌کنیم. دنباله‌ی σ را به چند مرحله[۱۲] تقسیم می‌کنیم

$$\sigma = \sigma_۱, \ldots \quad [\sigma_{i+۱}, \ldots, \sigma_j], [\sigma_{j+۱}, \ldots, \sigma_l], \ldots$$

که هر مرحله دارای دقیقا k خطا است و در آخرین عضو آن هم خطا رخ داده‌است. برای مثال اولین مرحله با σ_j پایان می‌یابد که

$$j = \min\{t : \text{LRU has k page faults in } \sigma_{i+۱}, \ldots, \sigma_t\}$$

حالت هزینه‌ی یک مرحله را برای هر دوی LRU و MIN بررسی می‌کنیم. بنابر تعریف LRU ، k خطا دارد. نشان می‌دهیم که الگوریتم MIN در هر مرحله بایستی حداقل یک خطا داشته باشد. دو حالت متفاوت را بررسی می‌کنیم.

حالت ۱: در مرحله‌ی یکسان LRU دو بار برای یک بخش p خطا می‌کند. بنابراین مرحله به شکل زیر است:

$$\ldots, [\sigma_{i+۱}, \ldots, \sigma_{p_۱} = p, \ldots, \sigma_{p_۲} = p, \ldots, \sigma_j], \ldots$$

توجه کنید که پس از اولین خطا روی p، p به $M_۱$ آورده‌می‌شود و دوباره از $M_۱$ حذف می‌شود تنها در صورتی که p اخیرا از بقیه کمتر استفاده‌شده‌باشد. بنابراین اگر یک خطای دیگری روی p رخ دهد نشان می‌دهد که همه‌ی k بخش دیگر موجود در $M_۱$ که قبل از $\sigma_{p_۲}$ هستند، پس از اولین خطا روی p و قبل از درخواست دوم به p وارد $M_۱$ شده‌اند. بنابراین $k + ۱$ بخش مختلف در این مرحله درخواست‌شده‌اند. این یعنی MIN در هر مرحله حداقل یک خطا داشته‌باشد.

حالت ۲: LRU روی k بخش مختلف خطا می‌کند.

در اینجا دو زیر حالت را بررسی می‌کنیم. با توجه به اینکه آخرین خطایی (مثلا p) که قبل از شروع مرحله صورت گرفته‌است.

الف) در مرحله‌ی فعلی، روی p دوباره خطا رخ می‌دهد.

$$\sigma_i = p, [\sigma_{i+۱}, \ldots, p \ldots, \sigma_j]$$

این حالت بسیار شبیه به حالت ۱ است. قبل از اینکه دومین خطا روی p رخ دهد بایستی k درخواست به بخش‌های غیر از p داده‌شود، بنابراین در مرحله‌ی فعلی کلا $k+۱$ درخواست به بخش‌های متفاوت وجود دارد. بنابراین الگوریتم MIN در این مرحله هم حداقل یک خطا دارد.

[۶] Page Fault
[۷] Least recently used
[۸] First-in First-out
[۹] Last-In-First-Out
[۱۰] Least-Frequently-Used
[۱۱] Configuration
[۱۲] Phase

ب) در مرحله‌ی فعلی هیچ خطایی روی p وجود ندارد.

$$\sigma_i = p, [\sigma_{i+1}, \ldots, \sigma_j]$$

رفتار MIN را بررسی می‌کنیم. قبل از شروع مرحله بایستی p در M_1 موجود باشد. توجه شود که در طول مرحله‌ی فعلی k درخواست متفاوت می‌رسند که هیچ کدام از آن‌ها p نیستند. بنابراین برای جا دادن همه‌ی آن‌ها MIN باید p را حذف کند.

نشان دادیم در همه‌ی حالت‌ها برای هر مرحله $C_{\text{MIN}}(\text{phase}) \geq 1$. ولی راجع به خطاهای قبل از شروع اولین مرحله حرفی نزدیم. با توجه به اینکه پیکربندی اولیه برای هر دو الگوریتم را یکسان گرفته‌ایم اولین خطا برای هر دو باید یکسان باشد. □

دسته‌ی کلی‌تری از الگوریتم‌ها وجود دارد که همه k-رقابتی هستند.

علامت‌گذاری[13]. الگوریتم علامت‌گذاری دنباله‌ی درخواست‌ها را در چند مرحله پاسخ می‌دهد. در ابتدای هر مرحله همه‌ی بخش‌های حافظه بدون علامت هستند. هرگاه یک بخش لازم می‌شود آن بخش علامت‌دار می‌شود. هنگام بروز خطا یکی از بخش‌های حافظه که علامت‌دار نیست به طور دلخواه انتخاب شده و حذف می‌شود. یک مرحله تمام می‌شود وقتی که همه‌ی بخش‌های حافظه علامت‌دار هستند و یک خطا بروز کند. در این صورت همه‌ی علامت‌ها پاک می‌شود و یک مرحله‌ی جدید شروع می‌شود.

الگوریتم LRU در واقع یک نوع الگوریتم علامت‌گذاری است. به طور کلی استراتژی‌های علامت‌گذاری در [۲و۳] بررسی شده‌اند. تورینگ[14] [۵] نشان داد که هر الگوریتم علامت‌گذاری k-رقابتی است. در واقع الگوریتم‌های قطعی در بهترین حالت k-رقابتی هستند.

یک الگوریتم بهینه‌ی آفلاین برای مسئله‌ی صفحه‌بندی توسط بلادی[15] [۱] ارایه شده‌است. الگوریتم MIN نام دارد و به شکل زیر عمل می‌کند.

MIN : هنگام بروز خطا بخشی را حذف کن که در آینده‌ی دورتر از بقیه دوباره درخواست می‌شود.

بلادی نشان داد که روی هر دنباله از درخواست‌ها این الگوریتم کمترین تعداد خطا را دارد.

قضیه ۳. [۱] *الگوریتم MIN یک الگوریتم بهینه‌ی آفلاین برای مسئله‌ی صفحه‌بندی است.*

قضیه ۴. [۴] *الگوریتم قطعی آنلاین برای مسئله‌ی صفحه‌بندی وجود ندارد که ضریب رقابتی[16] آن کمتر از k باشد. به عبارت دیگر برای هر الگوریتم آنلاین A دنباله‌ای مانند $\sigma^A = \sigma_1^A \ldots \sigma_n^A$ وجود دارد به طوری که $C_A(\sigma^A) \geq k.C_{MIN}(\sigma^A)$. در واقع این دنباله را می‌توان از مجموعه‌های $k+1$ بخش انتخاب کرد.*

اثبات. فرض کنید σ_i^A بخشی باشد که از M_1 پس از پاسخ دادن به $\sigma_1^A \ldots \sigma_{i-1}^A$ حذف شده است.

لم ۵. *برای هر دنباله‌ی متناهی σ که از بین $k+1$ بخش انتخاب شده است داریم:*

$$C_{MIN}(\sigma) \leq \frac{|\sigma|}{k}$$

اثبات. فرض کنید σ_i یک خطا برای MIN به وجود می‌آورد. نشان می‌دهیم MIN روی هیچ کدام از $\sigma_{i+1}, \ldots, \sigma_{i+k-1}$ هیچ خطایی نخواهد داشت. فرض کنید p بخشی باشد که MIN برای پاسخ به σ_i حذف می‌کند. با توجه به اینکه دقیقا $k+1$ بخش داریم، خطای بعدی باید روی p رخ دهد. توجه شود که هر بخش دیگر موجود در M_1 باید قبل از درخواست بعدی p درخواست داده شود.(بنا بر تعریف p بعد از همه درخواست داده می‌شود.) بنابراین حداقل $k-1$ درخواست σ_i را از خطای بعدی جدا می‌کند. □

حال فرض کنید σ یک پیشوند σ^A به طول kl باشد. بنابراین $C_A(\sigma) = kl \geq k.C_{\text{MIN}}(\sigma)$. پس A در بهترین حالت k-رقابتی است. □

[13] Marking
[14] Toring
[15] Belady
[16] Competitive ratio

مراجع

[1] LA .Belady, *A study of replacement algorithms for virtual storage computers* , IBM Systems Journal 5:78–101, 1966.

[2] A. Borodin and S. Irani and P. Raghavan and B. Schieber , *Competitive paging with locality of reference* , Journal of Computer and System Sciences 50:244–258.

[3] A. Fiat and RM. Karp and LA. McGeoch and DD. Sleator and NE. Young , *Competitive paging algorithms* , Journal of Algorithms 12:685–699, 1991.

[4] DD. Sleator and RE. Tarjan , *Amortized efficiency of list update and paging rules* , Communications of the ACM 28:202–208, 1985.

[5] E. Trong , *A unified analysis of paging and caching* , Algorithmica 20:175– 200, 1998.