مباحث ویژه در سیستم‌های دیجیتال

نیم‌سال اول ۱۴۰۰-۱۴۰۱

مدرس: دکتر ایمان غلام‌پور

دانشکده مهندسی برق

تمرین سری اول

نام و نام‌خانوادگی: امین کشیری — شماره دانشجویی: ۹۷۱۰۱۰۲۶

تمرین ۱

آ) بردار v و ماتریس A را در نظر می‌گیریم. ابتدا بردار v را به تکه‌هایی تقسیم می‌کنیم که در RAM جای گیرند. فرض کنید تعداد این تکه‌ها c تا باشد و در نتیجه هر تکه‌ای از بردار v ، $s = \frac{n}{c}$ درایه دارد. ماتریس A را نیز به صورت ستونی به گروه‌هایی با عرض s ستون تقسیم می‌کنیم. به هر کدام از کامپیوترهای map ، بخشی از یکی از این ستون‌ها را می‌دهیم.
یعنی اندیس ستون‌های تمام درایه‌هایی که در یک کامپیوتر map قرار دارند، همه در یکی از این c گروه قرار می‌گیرد. الان اگر به هر کامپیوتر map تنها آن بخشی از بردار v که متناظر با درایه‌هایی که دارد است را بدهیم، می‌تواند تمام ضرب‌های مورد نیاز خود را حساب کند. پس الان می‌توانیم به این صورت عمل کنیم که هر درایه‌ی ماتریس A را در درایه‌ی متناظر v ضرب کنیم، و با کلید i به reducer شماره i بفرستیم. این همان راه حلی است که در کلاس به صورت شفاهی مطرح شد.
در این حالت خاص، داریم:

$\forall a_{ij} : < key = i, value = a_{ij} \cdot v_j >$

r (replication rate) $= 1$

q (reducer size) $= n$

تعداد گروه‌های reduce دقیقا برابر با n است اما محدودیتی روی تعداد گروه‌های map وجود ندارد و کافی است آنقدر زیاد باشد که به اندازه‌ی کافی همزمانی داشته باشیم. هزینه‌ی ارتباطات [1] نیز در این حالت برابر است با خواندن تمامی ورودی‌ها، و همچنین به ازای هر درایه از ماتریس A نیز یک کلید جدید می‌سازیم. پس در کل:

CC (Communication Cost) $= O((mn + n) + mn) = O(mn)$

از تحلیل بیشتر این حالت ساده می‌گذریم زیرا می‌خواهیم حالت کلی‌تر و در واقع طیف چنین الگوریتم‌هایی را تحلیل کنیم.

دقت کنید که از اول دلیل گروه گروه کردن ستون‌های A (و یا متناظرا سطرهای v) به این خاطر بود که آن قسمت مورد نیاز از v کاملا در RAM قرار بگیرد. اما نیازی نیست که تعداد reducer ها را نیاز متناظر با تعداد «این» گروه‌ها بگذاریم. در واقع در حالت کلی تر، به این صورت عمل می‌کنیم:

- هر l سطر را به عنوان یک گروه در نظر می‌گیریم. تعداد کل گروه‌هایی که برای سطرها تشکیل شده است برابر است با $U = \frac{m}{l}$.
- h را تابع hash ی در نظر بگیرید که سطر i را به گروه متناظرش مثلا u هش می‌کند. حال:

MAP

$\forall a_{ij} : < key = h(i) = u,\ value = (i, a_{ij} \cdot v_j) >$

- همانطوری که گفتیم تعداد کل گروه‌ها برابر است با U . در نتیجه تعداد کل reducer های مورد نیاز نیز U است. در هر reducer نیز به ازای هر سطری که در آن گروه است، n مقدار می‌آید. پس:

REDUCE

for all i in this group: get values that start with i; add value[1] to find b_i

[1] Communication Cost

حال با توجه به توضیحات بالا داریم:

$r = 1$
$q = n \cdot l = n \cdot \frac{m}{U}$
$CC = O((mn + n) + mn) = O(mn)$

همانطوری که می‌بینید با اضافه کردن مفهوم گروه‌ها، می‌توانیم تعداد reducer ها که همان U است را کاهش دهیم. اما این به قیمت افزایش q است. اما این کار تغییری در r و CC نداد.

دقت کنید که این روش، کرانی روی r به ما نمی‌دهد و در آن r ثابت است. اما برای q داریم:

$n \leq q \leq nm$

همچنین تعداد reducer ها نیز بین 1 و m می‌تواند متغیر باشد (اما همانطور که گفتیم تعداد mapper ها مستقل از الگوریتم ما است. البته طبیعتا کران پایین آن ۱ است، و کران بالای آن نیز می‌تواند به تعداد کل درایه‌های ماتریس باشد!).

ب) برای این قسمت، دو ایده‌ای اصلی در کلاس مطرح شد. ابتدا این دو ایده را تحلیل می‌کنیم، سپس روش یک مرحله‌ی را به صورت کلی تر تحلیل می‌کنیم (و به صورت طیف در می‌آوریم).

راه اول: همان راه دو مرحله‌ای بحث شده در کلاس. در این حالت، در مرحله‌ی اول داریم:

$r = 1$
$q = m + p$
$CC = O((mn + np) + (mn + np)) = O(mn + np)$

در مرحله‌ی دوم، در کل $m + p$ کلید داریم اما تعداد کلیدهای خروجی mp است. پس r ما برابر است با تقسیم این دو مقدار. به هر کدام از reducer ها نیز n^2 کلید وارد می‌شود پس در مرحله‌ی دوم:

$r = \frac{mp}{m+p}$
$q = n^2$
$CC = O((mn + np) + (mpn))$

راه دوم: این راه راه یک مرحله‌ای بحث شده در کلاس است. در این حالت، به صورت زیر عمل می‌کردیم:

MAP
$\forall a_{ij} : \forall k : 1 \leq k \leq p :< key = (i, k), value = (A, j, a_{ij}) >$
$\forall b_{ij} : \forall k : 1 \leq k \leq m :< key = (k, j), value = (B, i, b_{ij}) >$

REDUCE
$sum = 0$
$\forall k : 1 \leq k \leq n : sum = sum + (a_{ik} \cdot b_{kj})$

یعنی تمام درایه‌های لازم برای محاسبه‌ی درایه‌ی (i, j) را به reducer (i, j) می‌فرستیم و در آنجا درایه‌های متناظر را ضرب می‌کنیم و نتیجه‌ی همه را جمع می‌کنیم. همانطوری که در کلاس دیدیم، در این روش به ازای هر درایه از ماتریس A به تعداد ستون‌های ماتریس B کلید جدید درست می‌شود و به ازای هر درایه‌ی B به تعداد سطرهای A (به این صورت می‌توانید تفسیر کنید که هر درایه‌ی A در محاسبه‌ی تمام ستون‌های یک سطر از ماتریس نهایی تاثیر دارد و هر درایه‌ی B در محاسبه‌ی تمام سطرهای یک ستون از ماتریس نهایی). در نتیجه داریم:

$$r = \begin{cases} p & \forall a_{ij} \\ m & \forall b_{ij} \end{cases} \Rightarrow \bar{r} = \frac{mnp + npm}{mn + np} = \frac{2mp}{m + p}$$
$q = 2n$
$CC = O((mn + np) + (mnp + npm)) = O(mnp)$

راه سوم: حال ما راه یک مرحله‌ای مطرح را شده را در حالت کلی تر بررسی می‌کنیم. ایده این است که تعداد کل reducer ها را از mp کاهش دهیم. به همین دلیل، سطرهای ماتریس A و همچنین ستون‌های ماتریس B را گروه‌بندی می‌کنیم. فرض کنید که تعداد سطرهای ماتریس A را به U گروه و ستون‌های ماتریس B را به V گروه تقسیم کرده‌ایم. در این صورت تعداد کل reducer های ما برابر با UV است. مراحل MAP کردن بسیار شبیه به حالت قبل است، با این تفاوت که درایه‌های سطر i م ماتریس A را به گروه مربوطه می‌فرستیم. پس دو تابع h و g را تعریف می‌کنیم به صورتی که سطرها و ستون‌ها را به گروه مخصوص خود هش می‌کند. به صورت دقیق‌تر:

MAP
$\forall a_{ij} : \forall k : 1 \leq k \leq V :< key = (h(i) = u, k), value = (A, i, j, a_{ij}) >$
$\forall b_{ij} : \forall k : 1 \leq k \leq U :< key = (k, g(j) = v), value = (B, i, j, b_{ij}) >$

REDUCE (u,v)

```
for all i that h(i) = u:
for all j that g(j) = v:
   sum = 0
   L = all Avalue that start with A and Avalue[1] = i and sort with Avalue[2]
   R = all Bvalue that start with B and Bvalue[2] = j and sort with Bvalue[1]
      for all j in range (1,n):
         sum = sum + L[j] + R[j]
   output((i,j), sum)
```

این روش بسیار شبیه به روش دوم است، با این تفاوت که تعداد reducer ها را کاهش داده‌ایم. هر reducer نیز به جای محاسبه‌ی یک درایه از ماتریس نهایی، تعدادی از درایه‌ها را محاسبه می‌کند. حال پارامترهای این الگوریتم را محاسبه می‌کنیم.

$$r = \begin{cases} V & \forall a_{ij} \\ U & \forall b_{ij} \end{cases} \Rightarrow \bar{r} = \frac{mnV+npU}{mn+np} = \frac{mV+pU}{m+p}$$

$$q = n \cdot (\text{number of rows in each A group}) + n \cdot (\text{number of columns in each B group})$$
$$= n \cdot \frac{m}{U} + n \cdot \frac{p}{V} = n \cdot (\frac{m}{U} + \frac{p}{V})$$
$$CC = O((mn + np) + mnV + npU)$$

همانطور که می‌بینید در حالت حدی که U و V برابر با m و p باشند، تحلیل ما به حالت دوم که بررسی کرده بودیم میل می‌کند. همچنین با کاهش تعداد گروه‌ها، می‌توانیم r را پایین بیاوریم اما در عین حال q افزایش پیدا می‌کند. حال سعی می‌کنیم که کران‌هایی روی r و q پیدا کنیم. اگر در حالت کلی بگیریم: $V = U = X$ آنگاه با حل معادلات بالا داریم:

$$\bar{r} = X$$
$$q = \frac{n\cdot(m+p)}{X} = \frac{n\cdot(m+p)}{\bar{r}}$$

که همانطوری که انتظار داشتیم می‌توانیم رابطه‌ای معکوس بین r و q ببینیم. یک کران بالا برای r و q می‌تواند با استفاده از مقادیر حدی U و V به دست آید. در این صورت:

$$1 \leq r \leq \frac{2mp}{m+p}$$
$$2n \leq q \leq n \cdot (m+p)$$

که همانطوری که می‌بینید یک طرف حالت‌های حدی دقیقا روش دوم ما است (که در آن r بشینه می‌شود اما q کمینه می‌شود).

دقت کنید که تعداد reducer های ما نیز برابر با UV است که این یعنی تعداد آن‌ها بین 1 و mp است. اما بازهم الگوریتم ما کرانی روی تعداد mapper ها نمی‌گذارد و تعداد آن‌ها می‌تواند هر اندازه‌ی باشد (و یک حالت حدی آن می‌تواند به تعداد درایه‌های دو ماتریس باشد!).