به‌نام خداوند بخشنده و مهربان

تمرین سوم: نگاشت-کاهش
درس: پایگاه‌داده پیشرفته
نام و نام‌خانوادگی: سید عمید اسدالهی مجد
شماره دانشجویی: 4003614004

استاد: محمدعلی نعمت‌بخش
دستیار: امیر سرتیپی

- لینک گیت‌هاب: https://github.com/amidmajd/hadoop-wordCount-exercise

مقدمه

در این تمرین برای راه‌اندازی Hadoop با استفاده از داکر انجام شد و کدهای mapper و reducer با استفاده از زبان پایتون نوشته شده است.

راهنمای اجرا

ابتدا محفضه‌های داکر را با اجرای دستور docker-compose up -d درون پوشه hadoop-cluster-docker که شامل فایلdocker-compose.yml است، راه‌اندازی می‌کنیم و سپس دستورات ذکر شده در فایل README.md که در پوشه اصلی تمرین قرار دارد را، درون پوشه اصلی تمرین، به ترتیب اجرا می‌نماییم.

راه‌اندازی هدوپ (Hadoop)

ابتدا داکر را روی سیستم نصب می‌نماییم که از وبسایت داکر برای ویندوز قابل دانلود و نصب است. سپس با ساختن فایل docker-compose.yml که تمام محفضه‌ها (Container) را آدرس‌دهی می‌کند و می‌سازد کار را ادامه می‌دهیم. با اجرای دستور docker-compose up ابتدا با استفاده از Dockerfile های تعریف شده در پوشه‌های مربوط به هر محفضه استفاده docker image مربوطه را ساخته، سپس آن‌ها را با تنظیمات موجود در فایل docker-compose.yml اجرا می‌نماید و محفضه‌های مورد نیاز را می‌سازد.

هر یک از بخش‌های Hadoop یک محفضه مخصوص به خود با پورت جداگانه خود در داکر دارند که عبارتند از:

- Name Node : 9870
- Data Node : 9864
- Node Manager : 9864
- Resource Manager : 8088
- History Server : 8188

فایل docker-compose.yml طوری تنظیم شده است که ۴ نود داده (data node) داشته باشیم.

آدرس دسترسی به هریک از این محفضه‌ها در فایل README.md که در پوشه مربوط به داکر (Hadoop-cluster-docker) قرار دارد، نیز ذکر شده است. رابط کاربری تحت وب هریک از این محفضه‌ها نیز قابل دستری است، به طول مثال با رفتن به آدرس http://localhost:9870 می‌توان به NameNode دسترسی پیدا کرد. (شکل ۱)

Cluster ID:	CID-b183b175-9739-46bb-8580-209c584250cb
Block Pool ID:	BP-87608729-172.22.0.2-1637608490980

Summary

Security is off.

Safemode is off.

40 files and directories, 18 blocks (18 replicated blocks, 0 erasure coded block groups) = 58 total filesystem object(s).

Heap Memory used 105.01 MB of 252 MB Heap Memory. Max Heap Memory is 1.28 GB.

Non Heap Memory used 53.53 MB of 54.88 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	1003.93 GB
Configured Remote Capacity:	0 B
DFS Used:	1.53 GB (0.15%)
Non DFS Used:	14.58 GB
DFS Remaining:	936.56 GB (93.29%)
Block Pool Used:	1.53 GB (0.15%)
DataNodes usages% (Min/Median/Max/stdDev):	0.10% / 0.16% / 0.20% / 0.04%
Live Nodes	4 (Decommissioned: 0, In Maintenance: 0)

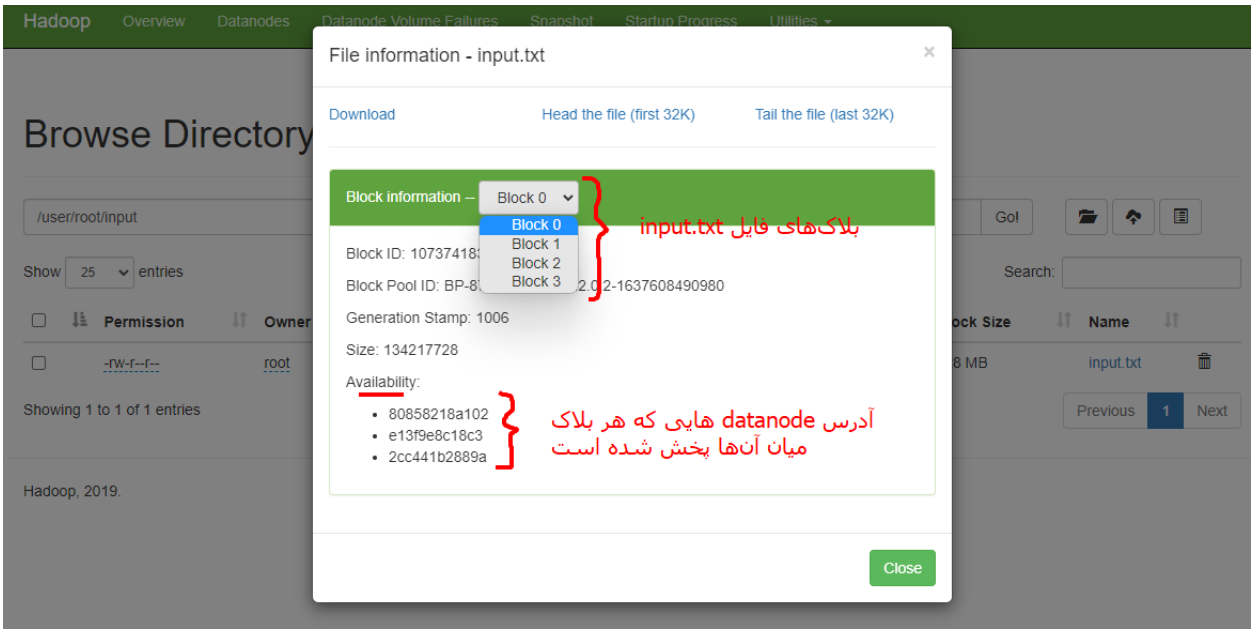
شکل 1: بخش Overview از رابط کاربری تحت وب NameNode

با استفاده از دستور docker exec -it *CONTAINER_NAME* /bin/bash نیز می‌توان به command line هر یک از این محفضه‌ها دسترسی پیدا کرد. همچنین با استفاده از دستور docker ps می‌توان از لیست محفظه‌ها و تنظیمات مهم آن‌ها را مشاهده کرد و از درستی اجرای هریک از محفضه‌ها اطمینان حاصل نمود. (شکل ۲)

```
CONTAINER ID   IMAGE                                     COMMAND                  CREATED          STATUS                    PORTS                                            NAMES
8ab3dbff0831   hadoop-cluster-docker_historyserver       "/entrypoint.sh /run…"   22 minutes ago   Up 22 minutes (healthy)   0.0.0.0:8188->8188/tcp, :::8188->8188/tcp        historyserver
d90b20382312   hadoop-cluster-docker_nodemanager1        "/entrypoint.sh /run…"   22 minutes ago   Up 22 minutes (healthy)   0.0.0.0:8042->8042/tcp, :::8042->8042/tcp        nodemanager1
de5a4423ce3a   hadoop-cluster-docker_resourcemanager     "/entrypoint.sh /run…"   22 minutes ago   Up 22 minutes (healthy)   0.0.0.0:8088->8088/tcp, :::8088->8088/tcp        resourcemanager
80858218a102   hadoop-cluster-docker_datanode3           "/entrypoint.sh /run…"   23 minutes ago   Up 23 minutes (healthy)   9864/tcp                                         datanode3
2cc441b2889a   hadoop-cluster-docker_datanode2           "/entrypoint.sh /run…"   23 minutes ago   Up 23 minutes (healthy)   9864/tcp                                         datanode2
c93bc1ee69e0   hadoop-cluster-docker_datanode4           "/entrypoint.sh /run…"   23 minutes ago   Up 23 minutes (healthy)   9864/tcp                                         datanode4
e13f9e8c18c3   hadoop-cluster-docker_datanode1           "/entrypoint.sh /run…"   23 minutes ago   Up 23 minutes (healthy)   9864/tcp                                         datanode1
aeb05ee42850   hadoop-cluster-docker_namenode            "/entrypoint.sh /run…"   23 minutes ago   Up 23 minutes (healthy)   0.0.0.0:9870->9870/tcp, :::9870->9870/tcp        namenode
```

شکل 2: خروجی دستور docker ps

یکی از بخش‌های کاربردی قابل دسترسی تحت وب NameNode بخش browse the file system است که از منوی utilites قابل دسترسی می‌باشد و می‌توان تمام فایل‌های موجود در HDFS را مشاهده نمود. به طور مثال فایل ورودی که حجمی در حدود ۵۰۰ مگابایت دارد در ۴ بلاک (هر بلاک حداکثر ۱۲۸ مگابایت) ذخیره شده است. این ۴ بلاک میان datanode ها پخش شده‌اند که در شکل ۳ قابل مشاهده است.



شکل 3: مشاهده فایل‌های موجود در HDFS و بلاک‌ها و پخش‌شدگی بلاک‌ها میان datanode ها

بارگذاری و یا دریافت ورودی و خروجی در HDFS

در شکل ۴ برای بارگذاری کدهای mapper و reducer موجود در پوشه src و همچنین فایل متنی ورودی بر روی HDFS، ابتدا آن‌ها را با استفاده از دستور docker cp از سیستم اصلی خودمان به پوشه home محفظه NameNode منتقل می‌کنیم.

سپس با استفاده از دستور docker exec -it namenode /bin/bash به command line محفظه NameNode وارد می‌شویم. حال با استفاده از دستور hdfs dfs یک دایرکتوی در HDFS ایجاد کرده و فایل متنی ورودی را به آن بر روی HDFS منتقل می‌نماییم. (hdfs dfs -put)

برای دریافت خروجی از HDFS از دستور hdfs dfs -get استفاده می‌کنیم و فایل متنی خروجی را به پوشه home محفظه NameNode منتقل می‌کنیم و در انتها با استفاده از دستور docker cp آن را به سیستم اصلی خودمان انتقال می‌دهیم.

Copying code & input Files

```
docker cp src/ namenode:/home/
docker cp input.txt namenode:/home/

hdfs dfs -mkdir -p input
hdfs dfs -put -f /home/input.txt input
```

Getting Output

```
hdfs dfs -get -f output/part-00000 /home/output.txt
docker cp namenode:/home/output.txt output.txt
```

شکل 4: بارگذاری و یا دریافت ورودی و خروجی در HDFS اجرا شده بر روی داکر

دستور Mapred در Hadoop

این دستور که در NameNode باید اجرا شود چند پارامتر به عنوان ورودی می‌گیرید و عملیات کاهش-نگاشت را با استفاده از mapper و reducer داده شده بر روی فایل متنی داده شده انجام داده و خروجی را در یک فایل متنی در آدرس داده‌شده ذخیره می‌کند. این دستور که در واقع یک فایل اجرایی جاوا می‌باشد تنها در صورتی قابلیت اجرای فایل‌های پایتون را دارد که پایتون در هریک از محفظه‌های داکر مربوط به Hadoop (یا بر روی سیستم اصلی که Hadoop در آن نصب شده است) نصب شده باشد.

فایل ورودی متنی باید حتما بر روی HDFS قرار داشته باشد و همچنین خروجی نیز بر روی HDFS ذخیره می‌شود.

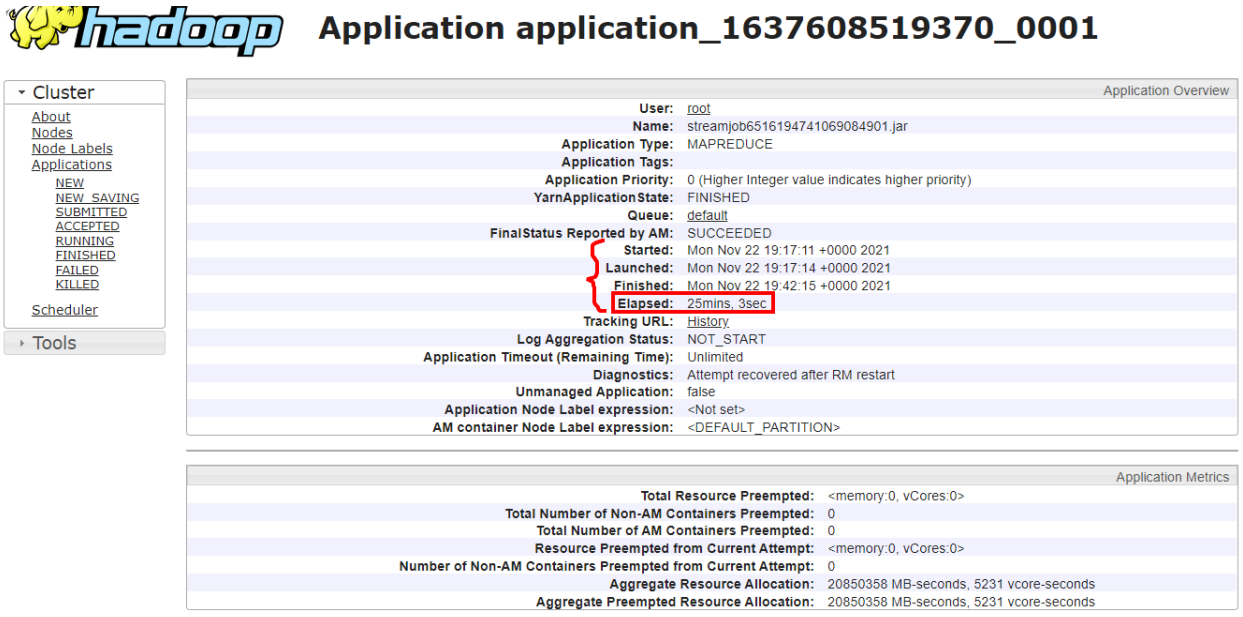
دستور اجرا در شکل ۵ قابل مشاهده است. در این دستور فایل‌های mapper.py و reducer.py به عنوان نگاشت‌هنده و کاهش‌دهنده داده شده و همچنین آدرس پوشه input موجود در HDFS که حاوی فایل‌های متنی ورودی است و آدرس پوشه خروجی روی HDFS نیز ذکر شده است.

Running MapReduce Job

```
mapred streaming -files /home/src/mapper.py,/home/src/reducer.py -mapper mapper.py -reducer reducer.py
-input input -output output
```

شکل 5: دستور اجرای عملیات نگاشت-کاهش

زمان اجرای کار، هم در command line پس از اتمام دستور اجرای کار، و هم در رابط کاربری تحت وب محفضه ResourceManager در آدرس http://localhost:8088 قابل مشاهده است. در شکل ۶ مشاهده می‌شود که زمان اجرای دستور شکل ۵ برابر با ۲۵ دقیقه و ۳ ثانیه می‌باشد.



شکل6: اطلاعات job اجرا شده در رابط کاربری تحت وب Resource Manager

عملیات map و reduce

دو فایل در پوشه src موجود است که مربوط به این عملیات می‌باشند. فایل ورودی متنی توسط Hadoop از طریق ورودی استاندارد (Standard Input) به هر بخش از عمیلات داده می‌شود و همچنین خروجی هر بخش نیز از خروجی استاندارد (Standard Output) توسط Hadoop دریافت می‌شود.

در کد برنامه از تولیدکننده‌های (Generators) پایتون استفاده شد که به‌جای اینکه ابتدا تمام عناصر را دریافت کنند و سپس یک لیست یا آرایه را برگردانند، به محض آماده شدن هر عنصر آن را برمی‌گردانند. در واقع هر عنصر در حافظه قرار نمیگیرد تا وقتی که بخواهیم آن را استفاده کنیم. در واقع اینگونه فقط زمان دسترسی به هر شئ (Object) آن شئ از محل اصلی (در اینجا ورودی استاندارد) دریافت شده و در حافظه قرار می‌گیرد. یعنی yield به جای return فقط اشاره‌گری به آن تکه از محل اصلی (در اینجا ورودی استاندارد) است. اینگونه می‌توان در مصرف حافظه بسیار صرفه‌جویی کرد و نیاز نیست تمام داده‌ها در حافظه بارگذاری شوند. همچنین در زمان اجرا نیز تاثیر بسزایی دارد زیرا نیاز نیست منتظر باشیم حلقه کامل انجام شود و سپس خروجی برگردانده شود و سپس توسط حلقه دیگری عملیات مورد نظر روی داده‌ها انجام شود.

Mapper.py

در این ماژول و تابع اصلی آن، خط به خط متن را از ورودی استاندارد با استفاده از تابع read_input می‌خوانیم و در هر خط روی هر کلمه تابع clean_word را اجرا می‌نماییم. اگر تابع clean_word کلمه‌ای را برگرداند آن‌را به همراه عدد یک در خروجی استاندارد چاپ می‌کنیم. یعنی این کلمه یک بار یافت شد. میان هر کلمه و تعداد آن با یک فاصله tab جدا شده است.

در تابع **read_input** از Generator ها استفاده شد و برای خواندن خط به خط از ورودی استاندارد، به محض دریافت هر خط از ورودی استاندارد اشاره‌گری به آن خط را باز می‌گرداند و منتظر نمی‌ماند تا تمام خط‌ها را در یک آرایه برگرداند. در واقع خروجی این تابع یک Iterable است.

تابع **clean_word** وظیفه تمیزکردن هر کلمه را دارد. ابتدا بررسی می‌کند و اگر کاراکتر نامربوطی (مانند نقطه، ویرگول، پرانتز و یا علامت نقل قول) در آن کلمه بود آن را با استفاده از تابع کمکی remove_chars از کلمه حذف می‌کند و کلمه را به حالت حروف کوچک (lowercase) تبدیل می‌کند. سپس بررسی می‌کند که آیا این کلمه یک کلمه توقفی یا حرف اضافه است یا خیر، در صورتی که کلمه یک کلمه توقفی یا حرف اضافه نبود خود کلمه و در غیر اینصورت پوچ (Null) برمی‌گرداند.

Reducer.py

در هدوپ به صورت خودکار یک مرحله میانی تحت عنوان shuffle & sort روی خروجی نهایی مرحله map انجام می‌شود و سپس به عنوان ورودی به مرحله reduce داده می‌شود. در این ماژول و در تابع اصلی آن، خط به خط متن از این ورودی ذکر شده توسط تابع read_mapper_output دریافت می‌شود. هر خط در واقع یک کلمه و تعداد آن (۱) است که با یک فاصله tab جدا شده‌اند. این کلمات به صورت الفبایی توسط shuffle & sort درونی Hadoop مرتب

شده‌اند. با استفاده از کتابخانه collections که یکی از کتابخانه‌های همراه پایتون است (جزؤ standard lib) و شئ Counter (در واقع یک دیکشنری از اشیاء و تعداد آن‌ها است و تعدای متد کاربردی نیز ارائه می‌کند) که توسط این کتابخانه ارائه می‌شود، تعداد کل هر کلمه محاسبه و ذخیره می‌شود. در انتها کلمات به صورت مرتب شده نزولی بر اساس تعدادشان در خروجی استاندارد چاپ می‌شوند. (۱۰ کلمه برتر درواقع ۱۰ کلمه اول خروجی reducer هستند)

در تابع **read_mapper_output** از Generator ها استفاده شد و دقیقا مشابه تابع read_input از ماژول mapper عمل می‌کند. تنها تفاوت این تابع این است که خط دریافتی را با استفاده از فاصله tab به دو بخش کلمه و تعداش تقسیم می‌کند و برمی‌گرداند.

۱۵ کلمه برتر به همراه تعداد تکرار قابل مشاهده در فایل خروجی output.txt

```
one 207333
time  153244
new 152315
also  151123
like  137818
get 120775
people  106396
use 102657
make  101616
first 98505
well  89473
work  84953
information 79597
see 78084
need  78074
```