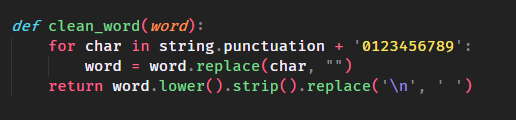
|  |
| --- |
| به‌نام خداوند بخشنده و مهربان  **تمرین سوم:** زمان اجرا **استاد:** محمد‌علی نعمت‌بخش  **درس:** تحلیل سیستم داده­های حجیم دستیاران: فاطمه‌ ابراهیمی، پریسا لطیفی، امیر سرتیپی  **نام و نام‌خانوادگی:** سید عمید اسدالهی مجد  **آدرس گیت:**<https://github.com/amidmajd/HW-3-hadoop-spark-comparison> |

**عملیات Map و Reduce در Hadoop**

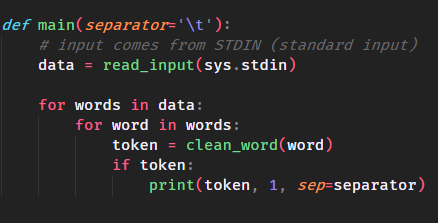
دو فایل **hadoop\_mapper.py** و **hadoop\_reducer.py** در پوشه src موجود است که مربوط به این عملیات می‌باشند. فایل ورودی متنی توسط Hadoop از طریق ورودی استاندارد (Standard Input) به هر بخش از عمیلات داده می‌شود و همچنین خروجی هر بخش نیز از خروجی استاندارد (Standard Output) توسط Hadoop دریافت می‌شود. این عمیات، با استفاده از قابلیت streaming در هدوپ انجام می‌شود که خروجی mapper را پس از مرتب‌سازی بر اساس کلمات، به عنوان ورودی به reducer می‌دهد. در هدوپ با بکارگیری streaming 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.

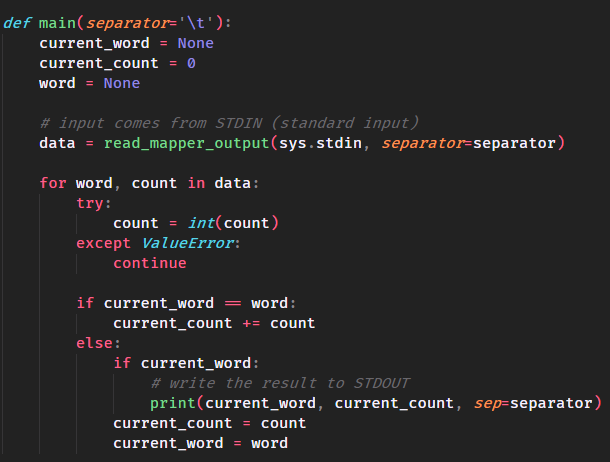
در این برنامه، دو تابع **read\_input** و **read\_mapper\_output** برای خواندن ورودی از تولید‌کننده‌های (Generators) پایتون استفاده شد که به‌جای اینکه ابتدا تمام عناصر را دریافت کنند و سپس یک لیست یا آرایه را برگردانند، به محض آماده شدن هر عنصر، آن را برمی‌گردانند. با استفاده از این روش به‌جای خواندن معمولی آرایه، می‌توان در مصرف حافظه صرفه‌جویی کرد و نیاز نیست تمام داده‌ها در حافظه بارگذاری شوند و سپس برگردانده 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.

تابع **clean\_word** وظیفه تمیز‌کردن هر کلمه را دارد. این تابع علائم سجاوندی (مانند نقطه، ویرگول، پرانتز و یا علامت نقل قول) را حذف می‌کند و کلمه را به حالت حروف کوچک (lowercase) تبدیل می‌کند.



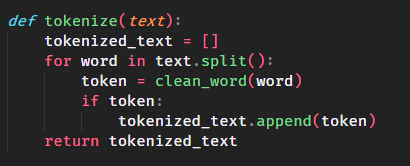
**توابع اصلی mapper و reducer**

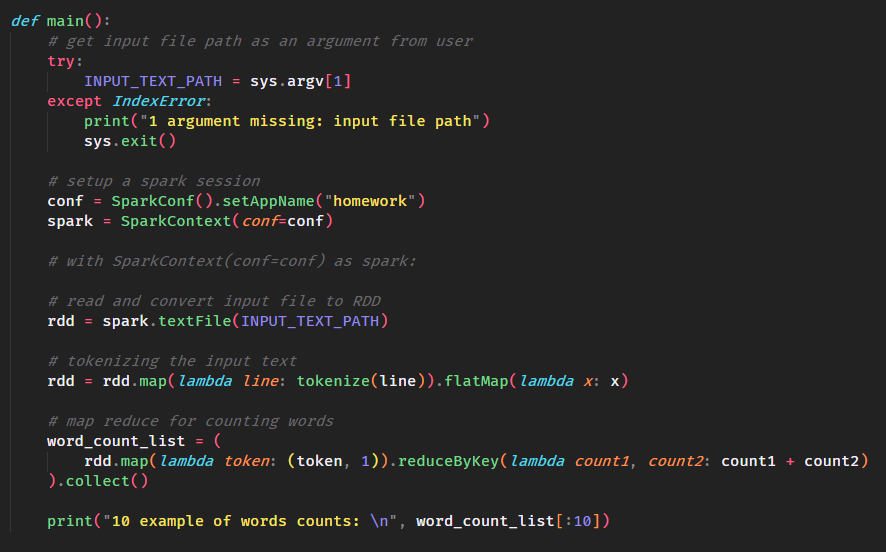




**عملیات Map و Reduce در Spark**

فایل spark\_map\_reduce.py عمیات نگاشت-کاهش را در اسپارک پیاده‌سازی می‌کند. در این فایل تابع tokenize وظیفه شکستن متن ورودی خود به توکن‌ها را برعهده دارد. در این بخش از برنامه پس از خواندن فایل متنی و تبدیل آن به یک RDD، با استفاده از تابع map اسپارک، ابتدا هر جمله‌ی درون RDD را به توکن‌های مختلف آن‌ تبدیل می‌کنیم و سپس با استفاده از تابع flatMap اسپارک، تمام این لیست‌های حاوی توکن را ادغام کرده و RDD را تبدیل به لیستی از توکن‌ها می‌نماییم. در انتها با استفاده از تابع map اسپارک، کلمات (توکن‌ها) را تبدیل به دوتایی‌های حاوی کلمه و عدد یک می‌نماییم (عمل نگاشت) و سپس با استفاده از تابع reduceByKey اسپارک، عملیات کاهش را پیاده‌سازی کرده و تعداد هر کلمه را بدست می‌آوریم.





مقایسه زمان اجرا در هدوپ و اسپارک

برای اجرای برنامه شمارش کلمه با استفاده از هدوپ و بررسی زمان اجرا از دستور زیر استفاده شد:

**time** **hadoop** **jar** $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar   
**-files** /home/amid\_asadollahi/src/hadoop\_mapper.py,/home/amid\_asadollahi/src/hadoop\_reducer.py   
**-mapper** hadoop\_mapper.py **-reducer** hadoop\_reducer.py **-input** file10G.txt **-output** output10G

برای اجرای برنامه شمارش کلمه با استفاده از اسپارک و بررسی زمان اجرا از دستور زیر استفاده شد:

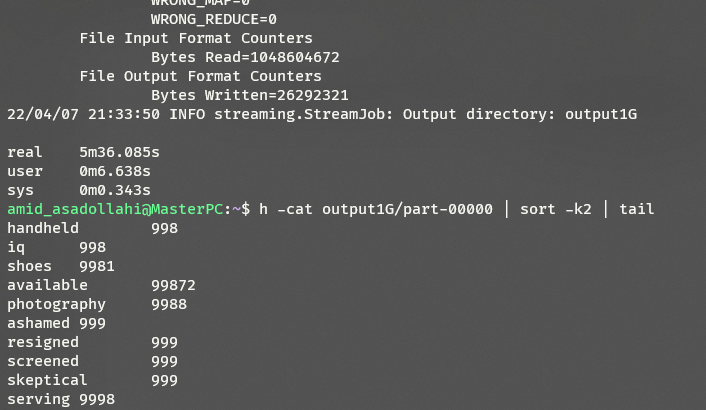
**time** **spark-submit** **--master** yarn **--num-executors** 100 **--deploy-mode client** ~/src/spark\_map\_reduce.py file10G.txt output10G.txt

برای اجرا در حالت cluster نیاز است تا فقط قسمت client از بخش deploy-mode بهcluster تغییر داده‌شود. هم‌چنین قابل ذکر است که بخش num-executors در دستور اسپارک به‌منظور تقسیم فایل و اجرای موازی درنظر گرفته شده‌است و عدد ۱۰۰ به‌معنای تقسیم فایل ورودی به ۱۰۰ بخش و اجرای این بخش‌ها به صورت موازی (تا جای ممکن که سیستم توان داشته‌باشد) می‌باشد. این عمل در هدوپ به‌صورت خودکار انجام می‌شود و split نام‌ دارد. بنابراین استفاده از num-executors برای مقایسه امری ضروری است.

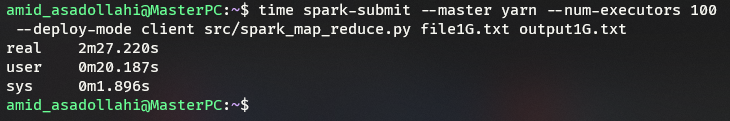
این دستورات با استفاده از هریک از فایل‌های ۱، ۵، ۱۰ و ۱۲ گیگابایتی اجرا شدند. برای اجرای نگاشت-کاهش برروی این فایل‌ها، ابتدا هریک از این فایل‌ها با دستور hdfs fs -put FILE\_NAME به فایل‌سیستم هدوپ منتقل شدند.

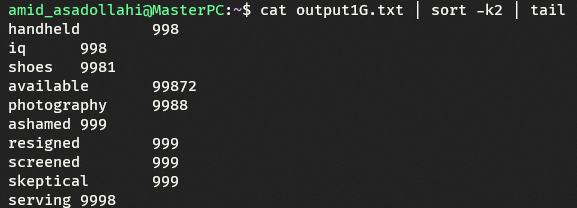
در هر مجموعه شکل ابتدا زمان اجرای برنامه نشان داده‌‌شده‌است و سپس برای نمایش خروجی آن دستور، ۱۰ کلمه با بیشترین تعداد تکرار پس از اجرای دستور نمایش داده‌شده‌است.

**اجرای نگاشت-کاهش برروی فایل ۱ گیگابایتی**

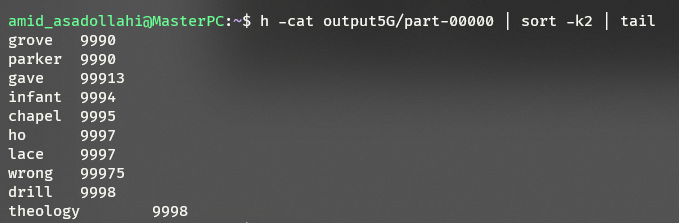
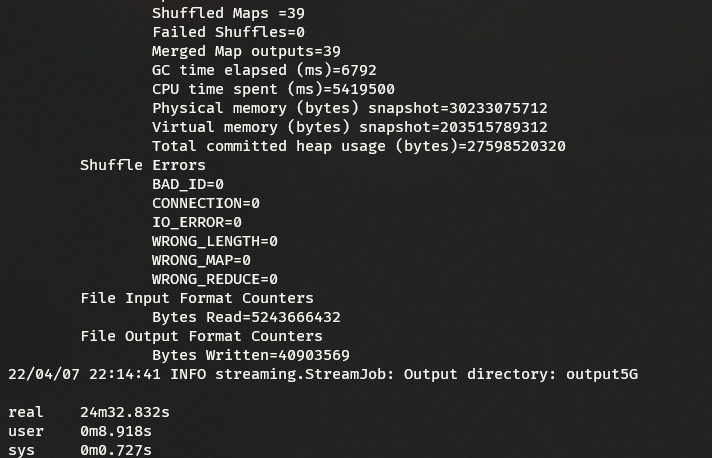
**هدوپ :**

**اسپارک:**

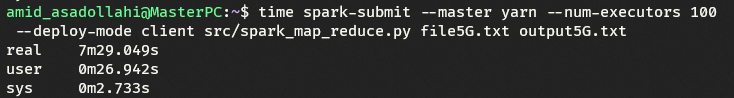


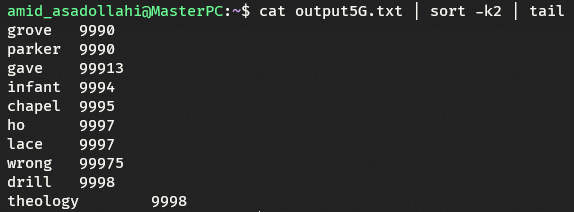


**اجرای نگاشت-کاهش برروی فایل ۵ گیگابایتی**

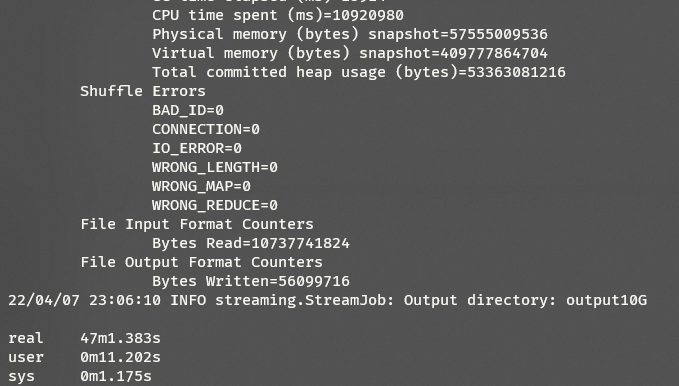
**هدوپ :**

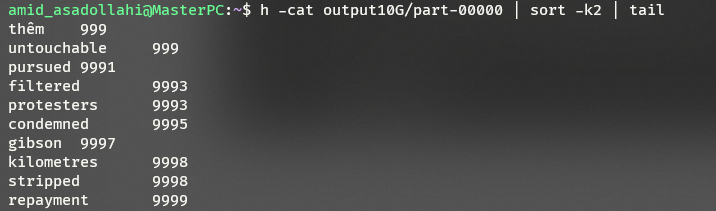
**اسپارک:**

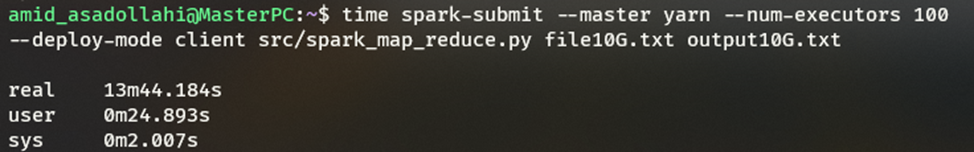


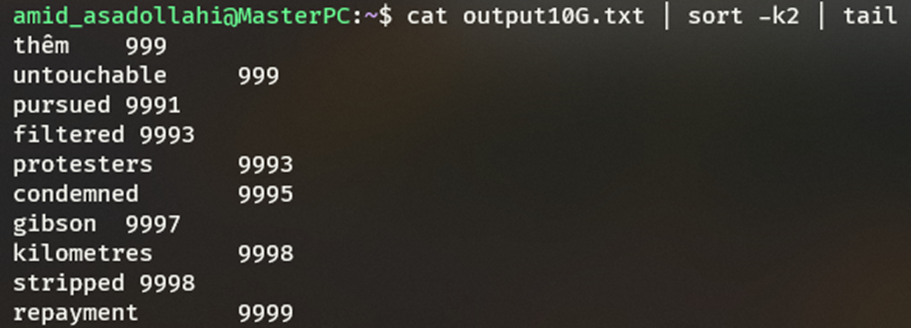


**اجرای نگاشت-کاهش برروی فایل ۱۰ گیگابایتی**

**هدوپ :** 

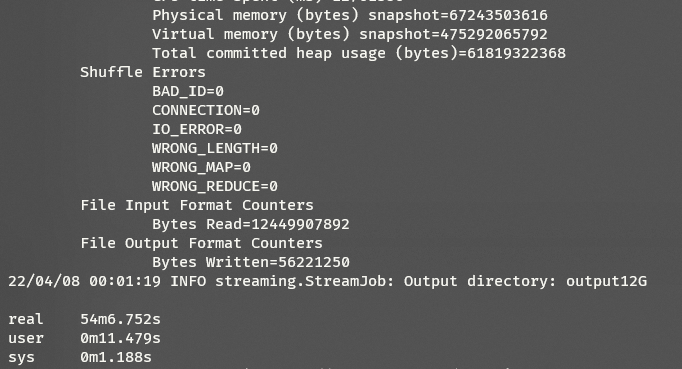


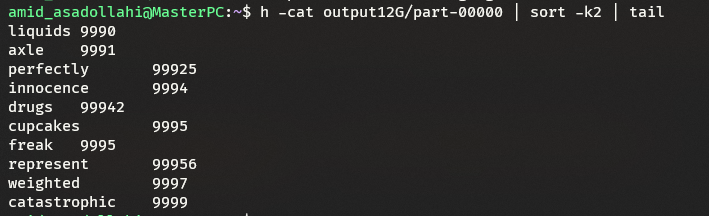
**اسپارک :**

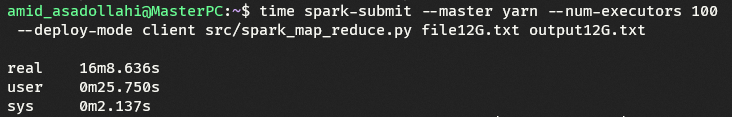


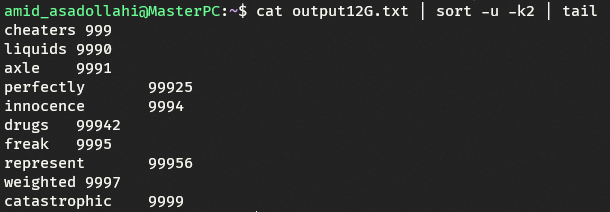
**اجرای نگاشت-کاهش برروی فایل ۱۲ گیگابایتی**

**هدوپ :**





**اسپارک:**

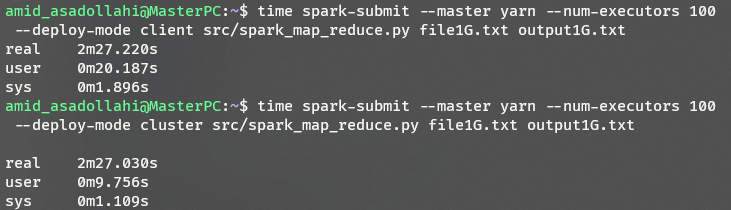


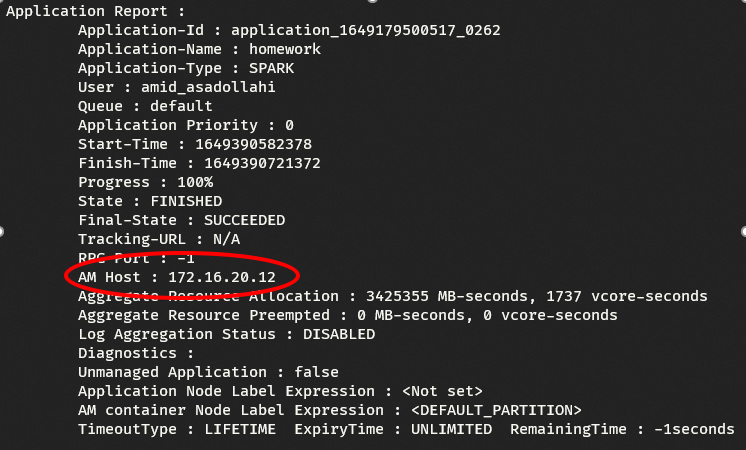
**مقایسه اجرای دستورات اسپارک در دو حالت Client و Cluster**

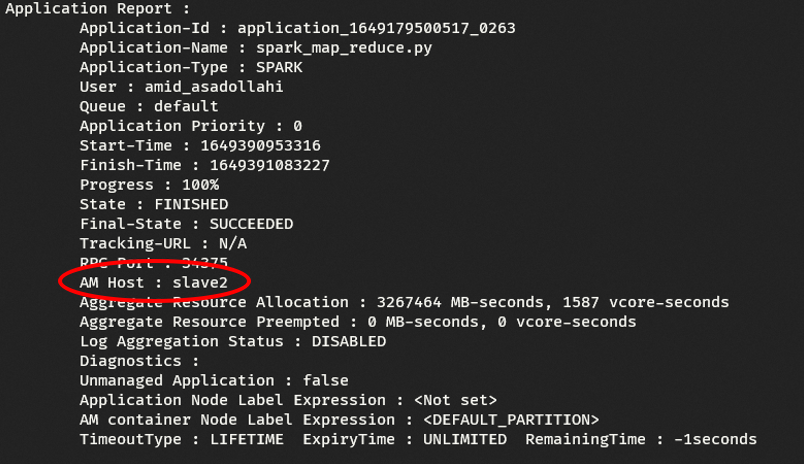
در حالت Client پس از اجرای دستور (Submit)، برنامه برروی همان ماشین اجرا می‌شود و به ماشین دیگری تخصیص داده نمی‌شود؛ بنابراین ماشین اجرا کننده (Client) باید تا زمان پایان اجرا روشن باشد و منابع خود را به اجرا اختصاص دهد. اما در حالت Cluster پس از اجرای دستور، برنامه توسط مدیر منابع اسپارک (Spark Resource Manager) به یکی از ماشین‌های Slave تخصیص داده می‌شود و دیگر نیازی نیست تا سیستم فراخواننده دستور روشن بماند و یا منابع خود را به اجرای دستور اختصاص دهد و فقط پس از پایان اجرا، به سیستم فراخواننده اطلاع داده می‌شود که اجرا خاتمه یافته‌است. درواقع در حالت Cluster مدریت نحوه اجرای دستورات را به مدیر منابع اسپارک داده‌ می‌شود تا دستور را برروی سیستم دارای بار کاری کمتر اجرا کند و اینگونه اجرا‌ها را در تمام سیستم‌ها پخش نماید.

در ادامه دو نمونه از این تفاوت نمایش داده ‌شده‌است. همانطور که مشاهده می‌شود در زمان اجرا تفاوتی ایجاد نشده است (به دلیل مشابهت تقریبی سیستم‌ها). تفاوت اصلی در محل اجرای دستور است؛ در حالت Client دستور برروی همان سیستم اجرا می‌شود ولی در حالت Cluster اجرای دستور به یکی از سیستم‌های Slave واگذار شده‌است.

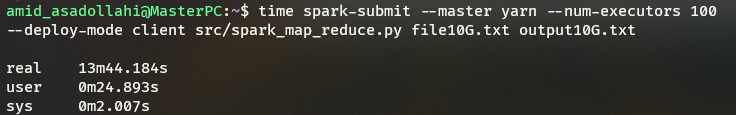
**اجرای دستور اسپارک برروی فایل ۱ گیگابایتی در حالت Cluster**







**اجرای دستور اسپارک برروی فایل ۱۰ گیگابایتی در حالت Cluster**

**

