علی بازشوشتری ۹۹۲۳۰۰۸

گزارش تمرین سوم رایانش ابری

بخش اول: MapReduce

توضیح الگوریتم Dijkstra

برای هر راس گراف یک فیلد distance داریم که نشان دهنده فاصله مبدا تا آن راس می باشد. در هر تکرار از اجرای map، جمع این فاصله با وزنی که میان راس مدنظر تا همسایش وجود دارد را به عنوان value به همسایش map میکنیم. پس بدین شکل داده هایی تولید می شوند که هر راس را به یک مقدار map میکنند و این مقدار نشان دهنده فاصله (distance) مبدا تا آن راس از طریق یکی از مسیرها می باشد. واضح است که ممکن است از مبدا تا هر راسی، چندین مسیر وجود داشته باشد.

پس از آن اقدام به shuffle کردن میکنیم و تمامی key-valueهایی که keyآنها (راس گراف) یکی است را در کنار یکدیگر نگه میداریم. در واقع این pairها نشان دهنده مسیرهای مختلف می باشند که از مبدا تا آن راس داریم.

پس از آن در مرحله reduce اقدام به min گرفتن از valueها که همان distanceها هستند می کنیم. با این کار از میان مسیرهای موجود، کوتاه ترین مسیر را نگه میداریم.

این عملیات ها را آنقدر تکرار میکنیم تا مقادیر (distanceها) همگرا شوند.

توضیح الگوریتم PageRank

$$PR(A) = (1-d) + d \times \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \ldots + \frac{PR(N)}{L(N)} \right)$$

$$P(n) = \alpha \left(\frac{1}{|G|} \right) + (1-\alpha) \sum_{m \in L(n)} \frac{P(m)}{C(m)}$$

در فاز map مطابق با الگوریتم pageRank با توجه به داده های هر گره که شامل pageRank فعلی آن و همچنین همسایه هایش می باشد، valueای که برای همسایه های گره مدنظر تولید می شود را محاسبه می کنیم(با تقسیم pageRank بر تعداد همسایه هایش) و value بدست آمده را به همسایه ها map میکنیم. این کار را در تمامی نودهای کلاستر و برای تمامی گره های گراف انجام میدهیم.

پس از آن اقدام به shuffle کردن میکنیم و تمامی key-valueهایی که keyآنها (راس گراف) یکی است را در کنار یکدیگر نگه میداریم.

پس از آن در مرحله reduce اقدام به جمع کردن valueهای مرتبط با یک key (راس گراف) میکنیم.در نهایت آن را در damping factor ضرب کرده و با (۱ – damping factor) یا $\frac{1-damping\ factor}{number\ of\ nodes}$ جمع میکنیم. Damping factor در واقع نشان دهنده احتمال آن است که به صورت رندوم از یک راس (وب پیج) به راس دیگری که همسایه است برویم. ۱ – damping factor احتمال آن است که به صورت رندوم به یک راس (وب پیج) دیگر برویم که الزاما همسایه راس ما نمی باشد.

بدین شکل مقدار جدید pageRank را برای هر راس گراف محاسبه میکنیم. این عملیات ها را آنقدر تکرار میکنیم تا مقادیر pageRank همگرا شوند.

علی بازشوشتری ۹۹۲۳۰۰۸

بخش دوم: Spark

کانتینرهای ساخته شده:

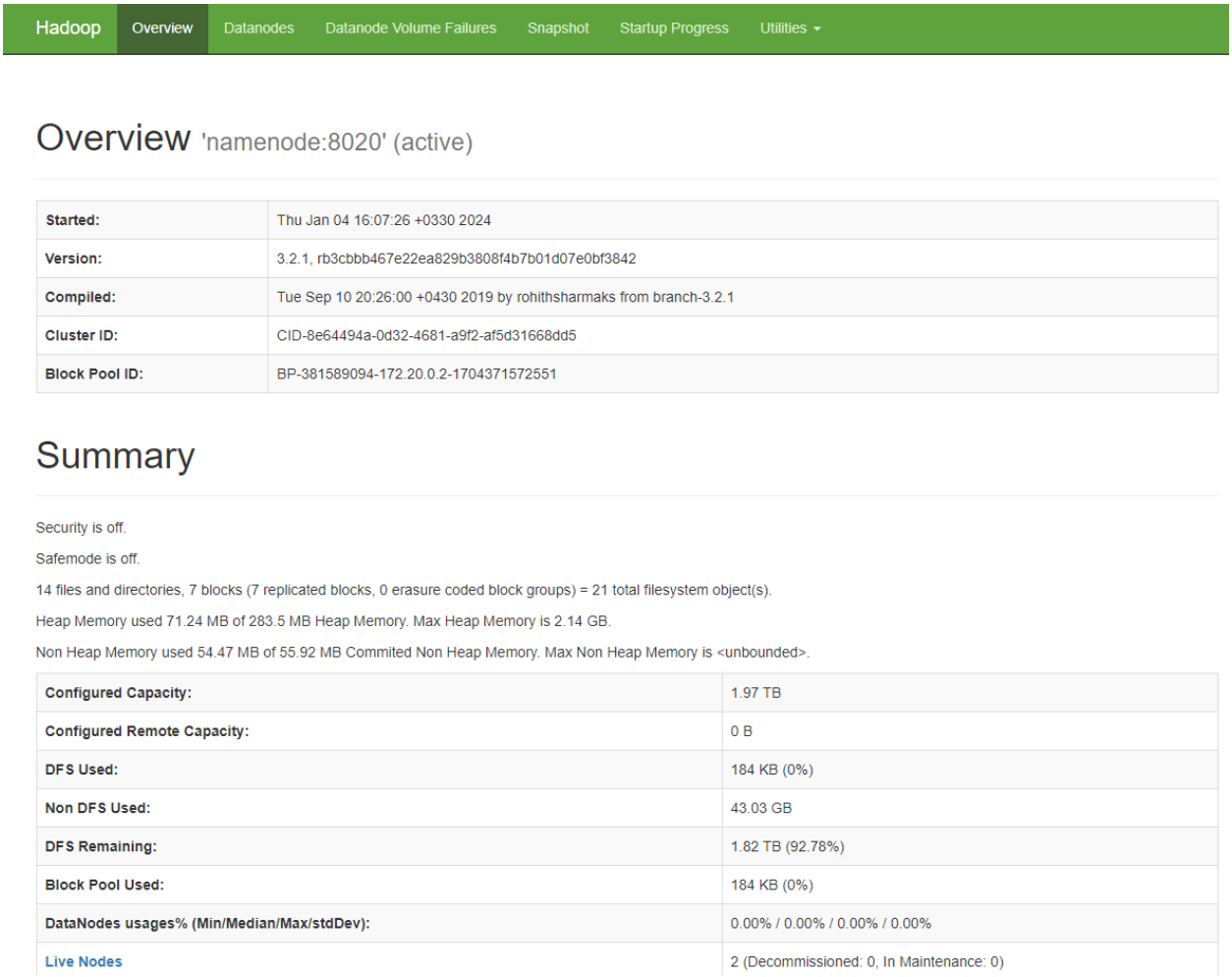
```
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS                    PORTS
                                          NAMES
68fac5f099a5   docker-resourcemanager     "/run.sh"                34 minutes ago   Up 29 minutes (healthy)   0.0.0.0:8089->8088/tcp, :::8089->8088/tcp
                                          resourcemanager
1fa82c7b27ef   docker-historyserver       "/run.sh"                34 minutes ago   Up 29 minutes (healthy)   0.0.0.0:8188->8188/tcp, :::8188->8188/tcp
                                          historyserver
f0f7b42d4720   docker-nodemanager1        "/run.sh"                34 minutes ago   Up 29 minutes (healthy)   0.0.0.0:8042->8042/tcp, :::8042->8042/tcp
                                          nodemanager1
332d1d8a49ba   docker-datanode2           "/run.sh"                34 minutes ago   Up 30 minutes (healthy)   9864/tcp
                                          datanode2
2a099a4d6d39   docker-datanode1           "/run.sh"                34 minutes ago   Up 30 minutes (healthy)   9864/tcp
                                          datanode1
cefcb3fbf27d   docker-namenode            "/run.sh"                34 minutes ago   Up 30 minutes (healthy)   0.0.0.0:8020->8020/tcp, :::8020->8020/tcp, 0.0.0.0:9870->9870/tcp, :::9870->987
0/tcp                                     namenode
557974e04530   docker-jupyter-notebook    "/bin/sh -c 'jupyter…"   34 minutes ago   Up 30 minutes             6066/tcp, 7077/tcp, 0.0.0.0:4040->4040/tcp, :::4040->4040/tcp, 0.0.0.0:8888->88
88/tcp, :::8888->8888/tcp, 8081/tcp       jupyter-notebook
1cba1f1d6b79   spark-base                 "/bin/sh -c ./start-…"   43 minutes ago   Up 30 minutes             6066/tcp, 7077/tcp, 0.0.0.0:7100->7000/tcp, :::7100->7000/tcp, 0.0.0.0:9091->80
81/tcp, :::9091->8081/tcp                 spark-worker1
6ff2cb68eff7   spark-base                 "/bin/sh -c ./start-…"   43 minutes ago   Up 30 minutes             6066/tcp, 7077/tcp, 0.0.0.0:7001->7000/tcp, :::7001->7000/tcp, 0.0.0.0:9092->80
81/tcp, :::9092->8081/tcp                 spark-worker2
5fb3c97936b6   spark-base                 "/bin/sh -c ./start-…"   43 minutes ago   Up 30 minutes             6066/tcp, 0.0.0.0:7077->7077/tcp, :::7077->7077/tcp, 0.0.0.0:9090->8081/tcp, ::
:9090->8081/tcp                           spark-master
```

توضیح وظیفه هرکدام از کانتینترهای Hadoop:

- Resourcemanager: در Hadoop، ResourceManager وظیفه مدیریت منابع موجود در یک کلاستر Hadoop را بر عهده دارد. این شامل تخصیص منابع به برنامه‌های مختلف (مانند برنامه‌های MapReduce) و نظارت بر اجرای آن‌ها است. پس به عبارتی مدیریت درخواست‌ها برای تخصیص منابع و برنامه‌ریزی و تخصیص دادن منابع به مقدار نیاز به نرم افزار های در حال اجرا روی کلاستر را انجام میدهد. ResourceManager همچنین با NodeManagers، که در هر گره از کلاستر قرار دارند، ارتباط برقرار می‌کند تا وضعیت منابع موجود در هر گره و وضعیت اجرای برنامه‌ها را پیگیری کند. این سیستم ارتباطی به ResourceManager امکان می‌دهد تا با داشتن دید کاملی از کل کلاستر، تصمیمات بهینه‌ای در مورد تخصیص منابع اتخاذ کند.
- Historyserver: نگهداری و ارائه اطلاعات jobهای MapReduce که به اتمام رسیده اند. این اطلاعات شامل جزییات عملکرد، وضعیت و خروجی jobها می باشد. همچنین logهای مربوط به این jobها را نیز نگهداری میکند.
- Nodemanager: هر node در کلاستر هدوپ یک nodemanager دارد که منابع آن node را مدیریت میکند. همچنین مسئول برقراری ارتباط با Resourcemanager، اجرای کانتینرهایی برای انجام تسک‌ها و نظارت بر عملکرد آن‌ها است.
- Datanode: در فایل سیستم HDFS وظیفه ذخیره سازی و مدیریت داده ها را به عهده دارد. این داده ها به صورت توزیع شده بر روی چندین node ذخیره می شوند. همچنین این کار سبب می شود که fault tolerance به ۰ برسد.

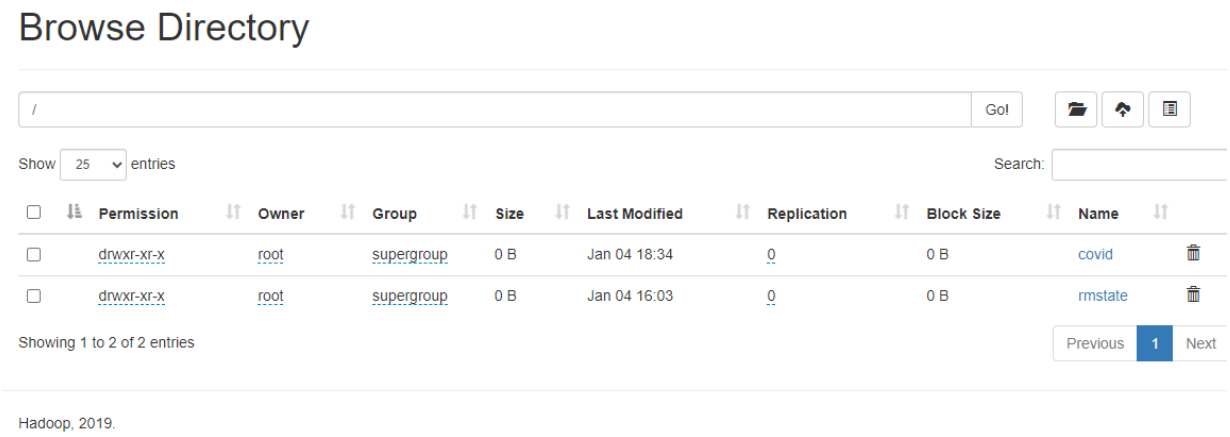
- Namenode: وظیفه نگهداری و مدیریت metadataهای مربوط به فایل های موجود در HDFS را دارد. همچنین single point of failure کلاستر هدوپ می باشد.

نمایش WebUI:



علی بازشوشتری ۹۹۲۳۰۰۸

نمایش فایل سیستم:



پاسخ به سوالات داخل نوتبوک:

- **shuffling**

Shuffling در واقع وسیله ای است برای جا به جایی و توزیع مجدد بلوک‌های داده میان executorها.

Shuffling در Apache Spark به فرآیند توزیع مجدد داده‌ها بین مختلف نودهای کلاستر به منظور اجرای محاسبات می‌پردازد. این فرآیند معمولاً زمانی اتفاق می‌افتد که نیاز به اجرای عملیات‌هایی مانند join، GroupBy یا Repartition بر روی دیتاست‌ها باشد. شافلینگ می‌تواند یکی از چالش‌برانگیزترین جنبه‌های توسعه برنامه‌های Spark باشد، زیرا می‌تواند منجر به افزایش زمان پردازش و استفاده از منابع شود.

نکته کلیدی در مورد شافلینگ این است که این فرآیند می‌تواند کند باشد، زیرا نیازمند حرکت داده‌ها بین نودهای مختلف است. این امر می‌تواند باعث ایجاد گلوگاه‌های شبکه شود و همچنین به زمان اضافی برای سریال سازی و دیسریال سازی داده‌ها نیاز دارد. به همین دلیل، بهینه‌سازی عملیات‌هایی که باعث شافل می‌شوند می‌تواند تأثیر قابل توجهی در عملکرد کلی برنامه‌های اسپارک داشته باشد.

برای کاهش هزینه‌های مربوط به شافلینگ، توسعه‌دهندگان می‌توان از استراتژی‌هایی مانند افزایش سطح موازی‌سازی، استفاده از عملیات‌های منطقی‌تر برای کاهش حجم داده‌هایی که نیاز به شافلینگ دارند، و بهینه‌سازی پارتیشن‌بندی داده‌ها استفاده کرد. همچنین، در نسخه‌های جدیدتر اسپارک، بهبودهایی در مدیریت حافظه و الگوریتم‌های شافلینگ ارائه شده است که می‌تواند کمک قابل توجهی به کاهش بار شافل کند.

۱. **تقسیم داده‌ها به چندین پارتیشن :(Partitioning)** در ابتدا، داده‌های ورودی به چندین پارتیشن تقسیم می‌شوند. هر پارتیشن می‌تواند در یک نود جداگانه پردازش شود.

۲. **اجرای عملیات ترانسفورمیشن :(Transformation)** ترانسفورمیشن‌هایی که نیازمند شافل هستند (مانند reduceByKey) بر روی هر پارتیشن اجرا می‌شوند. این فرآیند مقادیر کلید-مقدار را ایجاد می‌کند.

۳. **شافل داده‌ها** :داده‌های خروجی از ترانسفورمیشن‌ها بر اساس کلیدهایشان بین نودها منتقل می‌شوند. به این صورت، تمام داده‌هایی که دارای یک کلید مشترک هستند به یک پارتیشن یا نود خاص منتقل می‌شوند.

۴. **پردازش نهایی** :پس از انتقال، عملیات نهایی بر روی داده‌های شافل شده انجام می‌شود، مانند تجمیع یا مرتب‌سازی.

Sparkتسک‌هایی به نام ShuffleMapTask ایجاد می‌کند که هر کدام بخشی از داده‌ها را پردازش و آماده برای عملیات بعدی می‌کنند. پس از پردازش داده‌ها توسطShuffleMapTask ها، داده‌ها بر اساس کلیدهای مورد نیاز برای عملیات بعدی (مثل join یا groupBy) بین نودها منتقل می‌شوند. در نهایت، تسک‌هایی به نام ReduceTask یا دیگر تسک‌های ترکیبی داده‌های منتقل شده را پردازش نهایی می‌کنند.

- **Shuffle در application master UI**

درApache Spark ، Application Master UI یک رابط کاربری وب است که برای نظارت و مدیریت برنامه‌های Spark در حال اجرا استفاده می‌شود. این رابط کاربری اطلاعات مفیدی را درباره وضعیت اجرای برنامه، منابع مصرفی، متریک‌های عملکرد و جزئیات دیگر ارائه می‌دهد. به کمک Application master UI می توان جزییات مربوط به Shuffle را مشاهده نمود که سبب می شود درک بهتری از نحوه پردازش داده‌ها در کلاستر Spark و چالش‌های احتمالی مرتبط با Shuffle داشت.

- **DAG Scheduler در اسپارک چیست؟**

DAG Schedulerدر معماری اسپارک یک مؤلفه کلیدی است که برای مدیریت و برنامه‌ریزی وظایف (tasks)اجرایی در نودهای مختلف کلاستر به کار می‌رود. DAG مخفف Directed Acyclic Graph است که یک ساختار داده‌ای است برای نمایش جریان کارها یا وظایف با توجه به وابستگی‌های آن‌ها. Direct بودن آن به این دلیل است که عملیات ها در یک ترتیب مشخص اجرا می شوند. Acyclic بودن آن نیز به این معناست که در برنامه اجرایی دور نداریم. به صورت انتزاعی و سطح بالا، DAG نمایش دهنده برنامه اجرایی یک spark job است. به عبارتی از آن برای بازنمایی و بهینه سازی جریان عملیات ها در یک جاب پردازش داده استفاده می شود. در DAG، گره ها وظایف و تسک ها را نمایش می دهند و یال میان آنها نشان دهنده وابستگی میان هاست.

در اسپارک، هر برنامه کاربردی به یک سری از مراحل تقسیم می‌شود که خود این مراحل از یک سری وظایف تشکیل شده‌اند. DAG Scheduler نقشه‌ای از این مراحل و وابستگی‌های بین آن‌ها را می‌سازد. این ساختار به اسپارک کمک می‌کند تا تصمیم‌گیری‌های بهینه‌ای در مورد توزیع وظایف بر روی نودهای کلاستر و همچنین بازیابی خطاها انجام دهد.

در واقع DAG Scheduler مسئول ایجاد برنامه اجرایی برای یک جریان کاری است. این برنامه شامل اطلاعاتی درباره ترتیبی است که وظایف باید در آن اجرا شوند، و نیز داده‌هایی که باید بین این وظایف منتقل شوند. استفاده از DAG Scheduler به اسپارک این امکان را می‌دهد که بتواند به صورت موثرتری منابع را مدیریت کند و عملکرد کلی سیستم را بهبود ببخشد.

۱. **تقسیم برنامه به مراحل (Stages)** : اسپارک برنامه‌های کاربردی را به چندین مرحله تقسیم می‌کند. هر مرحله شامل گروهی از وظایف است که می‌توانند به صورت موازی اجرا شوند. تقسیم به مراحل بر اساس وابستگی‌های شافل (shuffle dependencies) انجام می‌شود.

۲. **ساختDAG :** پس از تعریف مراحل، DAG Scheduler یک DAG از این مراحل می‌سازد. این DAG وابستگی‌های بین مراحل را نشان می‌دهد.

۳. **برنامه‌ریزی وظایف** DAG Scheduler (**Task Scheduling**) برای هر مرحله، وظایف مورد نیاز برای اجرای آن مرحله را برنامه‌ریزی می‌کند. وظایف در مراحل مختلف بر اساس داده‌های مورد نیازشان تقسیم می‌شوند.

۴. **توزیع منابع** DAG Scheduler :منابع موجود در کلاستر را بررسی می‌کند و وظایف را به گونه‌ای توزیع می‌کند که به بهترین شکل از منابع استفاده شود.

۵. **بازیابی خطاها** :در صورت بروز خطا در اجرای یک وظیفه، DAG Scheduler می‌تواند آن وظیفه را در یک نود دیگر از کلاستر دوباره اجرا کند

ذکر این نکته نیز خالی از لطف نیست که می توان DAG را به کمک تکنیک هایی مانند pipelining ، caching و جا به جایی ترتیب taskها بهینه کرد تا عملکرد job افزایش یابد.