به نام خدا

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

تمرین سوم (فاز ۲) درس مبانی رایانش ابری

استاد درس: دکتر جوادی

نام : محمد حسین شهبازی

شماره دانشجویی:۴۰۰۳۱۰۸۸

سوال ۱

مقایسه از نظر سطوح انتزاع:

RDD: سطح پاييني از انتزاع را ارائه داده و مجموعهاي از اشيا توزيع شده هستند که به طور موازي در سرتاسر کلاستر، ذخيره ميشوند؛ همچنين امکان انجام عملياتهاي سادهاي مانند filter را ارائه ميدهند.

$DataFrames$:سطح بالاتري از انتزاع را ارائه داده و مشابه جداول در يك پايگاه داده رابطهاي هستند که به صورت توزيع شده و موازي ذخيره ميشوند؛ همچنين امکان انجام عملياتهايي مانندfilter ، group by، selectو ... را فراهم ميسازند.

مقايسه از نظر جنبه هاي عملكرد:

به دليل سطح پايينتر انتزاع، كنترل بيشتري بر روي دادهها و عملياتها دارند؛ همچنين عمليات بر روي RDD ها به صورت lazy evaluation انجام ميشود، يعني محاسبات تنها زماني اجرا ميشوند كه اكشنها فراخواني شوند.

$DataFrames$:از بهينهسازيهاي داخلي مانند optimizer catalyst بهرهمند ميشوند كه باعث بهبود عملكرد خودكار ميشود و در بيشتر موارد، سريعتر از RDDها هستند، زيرا بهينهسازيهاي داخلي بيشتري براي كاهش زمان اجرا دارند.

مقايسه از نظر موارد استفاده:

$RDDs$:براي عملياتهاي پيچيده و سفارشي كه نياز به كنترل دقيق دارند و الگوريتمهاي سنگيني مانند پردازش گراف و يادگيري ماشين، مناسب بوده و معمولا وقتي استفاده ميشوند كه لازم باشد تا به دادهها به صورت مستقيم و بدون استفاده از API سطح بالاتر دسترسي پيدا نمود.

DataFrames : براي عمليات تحليل دادههاي متداول كه ميتوانند به صورت SQLبيان شوند، مناسب بوده و براي بارگذاري، پاكسازي و تحليل دادههاي ساختاريافته و نيمه ساختاريافته استفاده ميشوند.

سوال ۲

از نظر قابلیت پرس و جو، SQL Sparkبه دلیل استفاده از زبان SQL که يك زبان استاندارد براي پايگاه داده است، كوئريهاي خواناتري را ارائه ميدهد و به راحتي ميتوان از آن براي كوئري زدن به منابع داده مختلف داده استفاده نمود؛ همچنين براي عملياتهاي پيچيده تحليل داده نيز مناسبتر ميباشد؛ اما API DataFrame قابليتهاي بيشتري براي تركيب عملياتهاي پيچيده داشته و اين امكان را ارائه ميدهد تا عملياتهاي سفارشي تري انجام گيرند، همچنين در زبانهاي برنامه نويسي مختلفي در دسترس بوده و براي برنامه نويسان، استفاده از آن راحت تر ميباشد . از نظر بهينه سازي عملكرد، هردو از optimizer catalyst براي بهينهسازي كوئريها و از execution tungsten engine براي بهبود عملكرد حافظه و CPUاستفاده ميكنند؛ با اين تفاوت كه درSQL Spark ، بهينه سازي به صورت خودكار انجام ميشود ولي درAPI DataFrame ، كاربران ميتوانند بهينه سازيهاي بيشتري را به صورت دستي انجام دهند .از نظر قابليت استفاده ازAPI ، SQL Sparkسادگي و خواناپي بيشتري داشته و چون زبان SQL استاندارد است، احتمال خطا كاهش ميابد؛ اما API DataFrame انعطاف پذيري و قابليت تركيب بيشتري دارد و ميتوان از توابع و ماژول هاي مختلف، براي ايجاد كوئريهاي قابل استفاده مجدد و سفارشي استفاده نمود .بهطور كلي SQL Spark مناسب براي تحليل داده و كساني است كه با SQL كار ميكنند و براي كوئريهاي ساده و سريع، ايدهآل ميباشد؛ اما API DataFrame براي توسعهدهندگاني كه به عمليات هاي پيچيده تر نياز دارند مناسب بوده و در پروژههايي كه نياز به تركيب چندين عمليات پيچيده است، استفاده ميشود.

سوال ۳

Partitioning در Spark Apache به فرآيند تقسيم دادهها به بخشهاي كوچكتر به نام پارتيشن ها اشاره دارد. هر پارتيشن يك بخش منطقي از داده است كه ميتواند به صورت مستقل پردازش شود؛ اين پارتيشن ها به صورت توزيع شده در ميان نودهاي مختلف يك كلاستر ذخيره ميشوند و پردازش موازي روي آنها انجام ميشود Partitioning .باعث افزايش كارايي و سرعت پردازش، بالانس بار و توزيع متوازن job ها، بهينهسازي حافظه و منابع، كاهش هزينه هاي ارتباطي و انعطاف پذيري و قابليت تحمل خطاي بيشتري ميشود.

سوال ۴

پارتیشن بندی پیش فرض (Default Partitioning)

Spark به صورت خودکار دادهها را بر اساس حجم و تعداد نودهاي موجود در کلاستر پارتیشن بندي میکند؛ این استراتژي براي بسیاري از کاربردها کافي است، اما در برخي موارد ممکن است نیاز به تنظیمات دستي باشد .در بسیاري از موارد، این پارتیشنبندي عملکرد خوبي دارد، اما ممکن است در مواردي که داده ها به صورت نامتوازن توزیع شدهاند یا حجم دادهها بسیار زیاد است، منجر به توزیع نامناسب بار شود.

پارتیشن بندی بر پایه کلید (Key-Based Partitioning)

در این روش، دادهها بر اساس کلید خاصي پارتیشن بندي میشوند؛ این استراتژي معمولا در عملیاتهاي join استفاده میشود زیرا داده هایي که داراي کلیدهاي مشابه هستند، در یك پارتیشن قرار میگیرند .این روش میتواند بهبود عملکرد قابل توج هي در عملیاتهاي join و by group داشته باشد، زیرا نیاز به انتقال دادهها بین نودها کاهش مییابد

پارتيشنبندي سفارشي (Custom Partitioning)

پارتيشن بندي سفارشي اين امكان را به كاربران ميدهد تا پارتيشن بندي خاصي را بر اساس نيازهاي خاص خود پيادهسازي كنند .اين روش ميتواند منجر به بهبود عملكرد در سناريوهاي خاص شود، زيرا كاربران ميتوانند دادهها را به نحوي پارتيشن بندي كنند كه منجر به كاهش انتقال داده ها و بهينه سازی استفاده از منابع شود.

بخش امتیازی۱:

مقایسه زمان محاسبه مجموع salary کمپاني ها با اندازهLarge ، با caching و بدون caching به صورت زیر میباشد.:

Bonus 1:

1.Calculating sum of salary with company size 'L' without caching.

```
rdd = salaries_df.rdd

start_time_no_cache = time.time()
dummy = rdd.filter(lambda row: row['company_size'] == 'L').aggregate(0,
                                                                     lambda acc, row: acc + row['salary'],
                                                                     lambda acc1, acc2: acc1 + acc2)

end_time_no_cache = time.time()
end_time_no_cache - start_time_no_cache
```

```
0.2176377773284912
```

2.Calculating sum of salary with company size 'L' with caching.

```
rdd_cached = rdd.cache()

start_time_cache = time.time()
dummy = rdd_cached.filter(lambda row: row['company_size'] == 'L').aggregate(0,
                                                                            lambda acc, row: acc + row['salary'],
                                                                            lambda acc1, acc2: acc1 + acc2)

end_time_cache = time.time()
end_time_cache - start_time_cache
```

```
0.3096776008605957
```

مشاهده میشود که با استفاده ازcaching ، سرعت پردازش بهتري به دست آمده است زیرا در پردازش دوم، به جاي اینکه داده ها مستقیم از منبع اصلي خوانده شوند، داده ها از حافظه خوانده شده اند.

بخش امتیازی ۲:

Narrow transformationها شامل عمليات هايي هستند كه در آنها هر پارتيشن از RDD خروجي فقط به يك پارتيشن از RDD ورودي وابسته است؛ اين تغييرات نياز به جابهجايي داده ها بين نودها ندارند و به راحتي در محل انجام ميشوند.

wide transformationها شامل عملياتهايي هستند كه در آنها داده ها بين نودهاي مختلف جابه جا ميشوند، به عبارت ديگر نياز به shuffle ميباشد؛ اين عملياتها به دليل نياز به بازتوزيع داده ها، باعث كاهش عملكرد ميشوند زيرا نياز به تبادل داده بين نودها افزايش مي يابد .مقايسه اي بين اين دو به صورت زير ميباشد:

Bonus 2:

1.Narrow transforamtion example.

[97]:
```
rdd = sc.parallelize(range(1, 1000000))

start_time_narrow = time.time()
narrow_result = rdd.map(lambda x: x * 2).collect()
end_time_narrow = time.time()
end_time_narrow - start_time_narrow
```

[97]:
```
0.42276501655578613
```

2.Wide transforamtion example.

[98]:
```
rdd_map = rdd.map(lambda x: (x % 100, x))

start_time_wide = time.time()
wide_result = rdd_map.groupByKey().mapValues(list).collect()
end_time_wide = time.time()
end_time_wide - start_time_wide
```

[98]:
```
0.953167200088501
```

مشاهده ميشود كه در transformation narrow زمان اجرا به دليل استفاده از map و عدم نياز به جابهجايي داده ها كوتاه است؛ ولي زمان اجراي transformation wide به دليل استفاده از key by وgroup نياز به shuffle و تبادل داده ها بين نودها، بيشتر ميباشد.